ANALYSIS OF RANDOMIZED SECURITY PROTOCOLS

BY

MATTHEW STEVEN BAUER

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Doctoral Committee:

Professor Mahesh Viswanathan, Chair
Professor Jose Meseguer
Assistant Professor Adam Bates
Assistant Professor Rohit Chadha

ABSTRACT

Formal analysis has a long and successful track record in the automated verification of security protocols. Techniques in this domain have converged around modeling protocols as non-deterministic processes that interact asynchronously through an adversarial environment controlled by a Dolev-Yao attacker. There are, however, a large class of protocols whose correctness relies on an explicit ability to model and reason about randomness. Lying at the heart of many widely adopted systems for anonymous communication, these protocols have so-far eluded automated verification techniques. The present work overcomes this long standing obstacle, providing the first framework analyzing randomized security protocols against Dolev-Yao attackers.

In this formalism, we present algorithms for model checking safety and indistinguishability properties of randomized security protocols. Our techniques are implemented in the Stochastic Protocol ANalyzer (SPAN) and evaluated on a new suite of benchmarks. Our benchmark examples include a brand new class of protocols that have never been subject of formal (symbolic) verification, including: mix-networks, dinning cryptographers networks, and several electronic voting protocols. During our analysis, we uncover previously unknown vulnerabilities in two popular electronic voting protocols from the literature.

The high overhead associated with verifying security protocols, in conjunction with the fact that protocols are rarely run in isolation, has created a demand for modular verification techniques. In our protocol analysis framework, we give a series of composition results for safety and indistinguishability properties of randomized security protocols.

Finally, we study the model checking problem for the probabilistic objects that lie at the heart of our protocol semantics. In particular, we present a novel technique that allows for the precise verification of probabilistic computation tree logic (PCTL) properties of discrete time Markov chains (DTMCs) and Markov decision processes (MDPs) at scale. Although our motivation comes from protocol analysis, the techniques further verification capabilities in many application areas.

To my wife and children, for their love and support.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Professor Mahesh Viswanathan. He was instrumental in harnessing my raw curiosity and creativity into powerful critical thinking and discovery. I would also like to thank Assistant Professor Rohit Chadha, who was a second advisor in all but formality. It was only through countless of hours of discussion with Mahesh and Rohit that I learned how to do effective and meaningful research. If not for them, this thesis would not have been possible.

I would like to thank Xizhong Zheng as well as his colleges Louis Friedler and Yanxia Jia at Arcadia University who first introduced me to research. Their support and individual dedication allowed me to spend the latter portion of my undergraduate studies largely exploring my own technical and research interests.

I would like to thank Giorgi Japaridze and Mirela Damian at Villanova University. I had many late night discussions with Giorgi through which I learned mathematical logic. These conversations helped foster numerous scientific breakthroughs early in my career.

I would like to thank my colleague and friend, Umang Mathur. It was our many conversations about science and life that are some of the fondest memories from my studies. His dedication to research inspired me to push the boundaries of my own creativity and endurance.

Last but not least, I would like to thank my parents Steve and Lori, my wife Lindsay and my children Nathan and Hannah.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Security protocols are highly intricate and vulnerable to design flaws. This has lead to significant effort in the construction of tools for the automated verification of protocol designs. Techniques for mechanized analysis of security protocols have converged around proving security in the *symbolic model* [1, 2, 3, 4, 5], where the assumption of perfect cryptography is made. Messages are symbolic terms modulo an equational theory (as opposed to bit-strings) and cryptographic operations are modeled via equations in the theory. Protocols are specified in (a variant of) the applied π-calculus [6] where the threat model is that of the *Dolev-Yao* attacker [7]. This omnipotent attacker has the ability to read, intercept, modify and replay all messages on public channels, remember the (potentially unbounded) communication history as well as non-deterministically inject its own messages into the network while remaining anonymous.

A growing number of security protocols are employing randomization to achieve privacy and anonymity guarantees. Randomization has become an essential component in protocols/systems for anonymous communication and web browsing such as Crowds [8], mix-networks [9], onion routers [10] and Tor [11] and has also been employed to achieve fair exchange [12, 13], vote privacy in electronic voting [14, 15, 16, 17] and denial of service prevention [18]. In the example below, we demonstrate how randomization can be used as a vehicle to achieve privacy in electronic voting systems.

Example 1.1 *Consider a simple electronic voting protocol comprised of* 2 *voters Alice and Bob, two candidates and an election authority. The protocol is as follows. Initially, the election authority will generate two private tokens t_A and t_B and send them to Alice and Bob encrypted under their respective public keys. These tokens will be used by the voters as proofs of their eligibility. After receiving a token, each voter sends his/her choice to the election authority along with the proof of eligibility encrypted under the public key of the election authority. Once all votes have been collected, the election authority tosses a fair coin. If tails turns up, it outputs Alice's vote followed by Bob's vote. In the event of a head, the order in which the votes are released is reversed. The security property of the protocol we are interested in is* vote privacy*, meaning an adversary should not be able deduce how each voter voted.*

All of the existing Dolev-Yao analysis tools are fundamentally limited to protocols that are purely non-deterministic, where non-determinism models concurrency as well as the interaction between protocol participants and their environment. There are currently no symbolic analysis tools allowing the protocol from Example 1.1 to be verified, a limitation

that has long been identified by the verification community. In the context of electronic voting protocols, [19] identifies three main classes of techniques for achieving vote privacy; blind signature schemes, homomorphic encryption and randomization. There the authors concede that protocols based on the latter technique are "hard to address with our methods that are purely non-deterministic." Catherine Meadows, in her summary of the over 30 year history of formal techniques in cryptographic protocol analysis [20, 21], identified the development of formal analysis techniques for anonymous communication systems, almost exclusively built using primitives with randomization, as a fundamental and still largely unsolved challenge. She writes, "it turned out to be difficult to develop formal models and analyses of large-scale anonymous communication. The main stumbling block is the threat model". Later, she goes on to add that the features of these systems are "harder to capture in a formal model, and even harder to analyze; thus research in this area has depended heavily on experimentation and simulation."

This work effectively breaks down the long standing barrier to automated verification of randomized security protocols, providing a mathematically rigorous foundation for modeling and mechanically analyzing the electronic voting protocols and anonymous communication systems described above. The breakthrough comes with a characterization of the subtle interaction between non-determinism and randomization. If the attacker is allowed to "observe" the results of private coin tosses in its scheduling decisions, then the analysis may reveal "security flaws" in correct protocols (see examples in [22, 23, 24, 25, 26, 27]). In Example 1.1, if the coin toss performed by the election authority is visible to the adversary, then protocol does not satisfy vote privacy. On the other hand, the attacker needs to remain powerful enough to, among other things, alter the order of protocol events and modify/forge messages on the network. The key insight in overcoming this problem is to analyze protocols with respect to attackers that are required to choose the same action in any two protocol executions it cannot distinguish. We propose trace-equivalence from the applied π-calculus [6] for the indistinguishability relation. In this framework, an attacker is a function from *traces*, the equivalence classes on executions under the trace-equivalence relation, to the set of attacker actions.

This thesis formalizes the preceding observations and introduces a powerful new framework for analyzing randomized security protocols. We capture the behavior of a randomized security protocol executed in the presence of a Dolev-Yao attack as a partially observable Markov decision process (POMDP). A POMDP is a state transition system that consists of both non-deterministic and probabilistic actions that are scheduled by an adversary who observes only a portion of the state information. Scheduling under partial information is what allows the model to capture private coin tosses. In this framework, we make the

following fundamental contributions.

- We introduce several algorithms for analyzing properties of randomized security protocols. Our techniques are implemented and evaluated a new protocol analysis tool.
- We present conditions under which properties of randomized security protocols are preserved through composition.
- We enhance state-of-the-art quantitative analysis tools for probabilistic systems through a new algorithm that allows for precise verification at scale.

The following sections introduce these topics in more detail.

1.1 AUTOMATED SYMBOLIC ANALYSIS OF SECURITY PROTOCOLS

In Chapter 4, we introduce a new tool, the Stochastic Protocol ANalyzer (SPAN), for analyzing randomized security protocols. SPAN analyzes two kinds properties: safety properties and indistinguishability properties. Properties of the former type allow one to prove the absence certain undesirable behaviors in a system. In Example 1.1, for instance, there should be no execution of the protocol in which the secret key of any of the protocol participants can be derived from the network traffic. On the other hand, indistinguishability is a property between two protocols. Essentially, two protocols are indistinguishable if the attacker cannot determine which of the two protocols he/she is interacting with. As observed by many authors [19, 28, 29, 30, 31], critical properties of security protocols such as anonymity, unlinkability, and privacy can be modeled using a notion of indistinguishability. Consider the protocol from Example 1.1, designed to preserve vote privacy. Such a property is achieved if the executions of the protocol in which Alice votes for candidate 1 and Bob votes for candidate 2 cannot be distinguished from the executions of the protocol in which Alice votes for candidate 2 and Bob votes for candidate 1.

There are a myriad of symbolic analysis tools that analyze safety properties of security protocols, for example, [2, 3, 5, 32, 33, 34, 35]. Indistinguishability properties are intrinsically more difficult to analyze, only recently have a few tools began to emerge [4, 36, 37, 38]. Unfortunately, none of the preceding tools have any support for dealing with randomization. As a result, most analysis efforts using these tools simply "abstract away" essential protocol components that utilize randomization, such as anonymous channels. For example, such an abstraction is made in [19, 39] where the authors conduct a symbolic analysis of the Fujioka, Okamoto and Ohta (FOO) electronic voting protocol. As we show in Section 3.4, this kind of abstraction can result in the incorrect design and analysis of a protocol.

There have been previous attempts to integrate randomization into symbolic protocol analysis. For example, [40] introduced a process calculus allowing operations for both non-deterministic and probabilistic choice. Unfortunately, the calculus did not capture many important properties of the threat model, such as the ability for protocol participants to make private coin tosses. As a result, properties of these processes are required to be formulated through a notion of bisumlation too strong to capture many natural properties. By contrast, our framework and analysis techniques have proven successful in verifying (and finding bugs in) a number of widely studied protocols. Using SPAN, we have conducted the first automated symbolic analysis mix-networks [9], dining-cryptographers-networks (DC-nets) [41], 3-ballot electronic voting [17, 42], the FOO electronic voting protocol [43], and Prêt à Voter [14]. During our analysis, we uncovered previously unknown bugs in the latter two protocols.

1.2 COMPOSITIONAL VERIFICATION OF SECURITY PROTOCOLS

Algorithms for symbolic protocol analysis solve inherently difficult computational problems. As the size of a protocol grows, so too does the computational burden for verification. This, in conjunction with the fact that security protocols are rarely run in isolation, has led to a deep interest in modular techniques for protocol analysis. The goal of such techniques is to identify sufficient conditions under which two protocols, proven secure in isolation, can be shown to be secure in an environment where they interact. Composition results of this form provide two main benefits. They help improve protocol designs by demonstrating how the interaction with other protocols can break security. And, they allow large protocols to be verified by decomposing them into smaller parts and stitching these smaller verification efforts together with compositional guarantees.

Protocols can interact either in a concurrent or sequential fashion. The latter being quite common in a number of protocols that run an initial sub-protocol to establish short-term secret communication keys. Notice that this scenario requires the key-establishment sub-protocol to share data with later phases of the protocol. Data sharing between composed protocols is particularly difficult to reason about in a compositional setting. Nonetheless, we are able to achieve powerful composition results for both inditinguishability and safety properties. Our composition framework provides a generic mechanism in which protocols can interact concurrently or sequentially as well as share secret data. Composition results for indistinguishability properties are given in Chapter 5 and composition of sate-based safety properties is considered in Chapter 6.

Our composition results for randomized protocols compliment the array modular analysis

techniques that have surfaced for the non-randomized case. Safety properties are considered in [44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56] and indistinguishability properties in [57, 56]. In [44, 45, 46] a general framework is proposed for proving that protocols compose securely. Other papers [48, 47] essentially show that protocol compositions are secure if messages from one protocol cannot be confused with messages from another protocol. This can be ensured if certain protocol transformations are made (see for example [49, 50, 52, 57]). Essentially, these protocol transformations require that all protocol messages are *tagged* with the protocol name and protocol instance to which they belong. [58] shows that this continues to be the case even when dishonest participants do not tag their messages properly. The exact choice of tagging scheme depends on the desired security property; incorrect tagging can actually make a secure protocol insecure [52]. In the computational model, the problem of composing protocols securely has been studied in [59, 60]. Our results are most closely related to [57, 56].

1.3 QUANTITATIVE MODEL CHECKING

Prior to our framework, techniques for analyzing security protocols have remained largely disjoint from techniques for analyzing systems with randomization. Many have attempted to use probabilistic model checkers such as PRISM [61], STORM [62] and APEX [63] to verify protocols that explicitly employ randomization. For example, [64] conducts an analysis of the Crowds anonymous web browsing protocol in PRISM. There they showed that system reconfigurations necessitated by users joining and leaving the system can lead to a degradation in security over time. In [65], an analysis of Chaum's Dining cryptographers protocol [41] was carried out the in CSP framework [66]. All of the works in this arena are ad-hoc in nature and don't provide a general verification mechanism. Furthermore, many simplifications are made to make modeling and analysis possible. The analysis from [64] considers only a passive adversary that observes but does not modify network traffic.

In this work, we are interested in the study of quantitative modeling checking techniques for two reasons. The analysis of probabilistic models remains a useful tool for analyzing certain aspects of security protocols. Furthermore, in many cases, analysis of randomized security protocols in our framework reduces to analyzing such models. Unfortunately, in order to scale to large models, state-of-the-art quantitative model checkers typically implement approximation techniques on top of floating-point arithmetic. As a result, there is no guarantee on the quality of the solution produced by these tools. This can lead to a completely incorrect logical analysis of a system or security property. To rectify this, we introduce a novel model checking technique that leverages the approximate solutions generated by these

model checking tools to compute exact solutions. Our technique incurs very little overhead and has been shown to work on a wide array of examples. The full details are given in Chapter 7.

1.4 OUTLINE

The remainder of this thesis is structured as follows. In Chapter 2, we define the mathematical preliminaries need for modeling and analyzing cryptographic protocols with randomization. In Chapter 3, we describe several randomized protocols that will serve as running examples in our composition results and benchmarks for our automated analysis tool. Chapter 4 describes the (SPAN). Chapters 5 and 6 detail our composition results for indistinguishability and reachability properties, respectively. In Chapter 7, we present our technique and tool for performing exact quantitative model checking at scale. We conclude with potential future work Chapter 8.

CHAPTER 2: PRELIMINARIES

2.1 PROBABILITY SPACES

Given a set Ω, $\Sigma \subseteq 2^{\Omega}$ is said to be a σ-algebra on Ω if Σ contains $\emptyset$ and is closed under complementation and countable union. A function $\mu : \Sigma \to [0,1]$ is said to be *countably additive* if for each countable collection $F_1, F_2, \ldots \in \Sigma$ of pairwise disjoint sets, $\mu(\bigcup_{i=1}^{\infty} F_i) = \sum_{i=1}^{\infty} \mu(F_i)$. A (sub)-probability space is a tuple (Ω, Σ, μ) where Ω is a set of events, Σ is a σ-algebra on Ω and $\mu : \Sigma \to [0,1]$ is a countably additive function such that $\mu(\emptyset) = 0$ and $\mu(\Omega) \leq 1$. The set Σ is said to be the set of events and μ the (sub)-probability measure of Ω. For $F \in \Sigma$, the quantity $\mu(F)$ is said to be the probability of the event F. If $\mu(\Omega) = 1$ then we call μ a probability measure.

A (sub)-probability space (Ω, Σ, μ) is said to be *discrete* if $\Sigma = 2^{\Omega}$. In a discrete (sub)-probability space, for any set $F \subseteq \Omega$, we have $\mu(F) = \sum_{x \in F} \mu(\{x\})$. Hence, for discrete (sub)-probability spaces, μ is completely determined by its value on singleton sets and we will consider μ a function from Ω to $[0,1]$. For a probability measure μ over Ω, let $\mathsf{support}(\mu) = \{x \in \Omega \,|\, \mu(x) > 0\}$. The set of all discrete probability distributions over Ω will be denoted by $\mathsf{Dist}(\Omega)$. Given any $x \in \Omega$, the *Dirac measure* on Ω, denoted δ_x, is the discrete probability measure μ such that $\mu(x) = 1$. Given two (sub)-probability measures μ_1 and μ_2 on a measure space (Ω, Σ) as well as a rational number $p \in [0,1]$, the convex combination $\mu_1 +_p \mu_2$ is the (sub)-probability measure μ such that for each set $F \in \Sigma$ we have $\mu(F) = p \cdot \mu_1(F) + (1 - p) \cdot \mu_2(F)$.

2.2 DISCRETE TIME MARKOV CHAINS

A discrete time Markov chain (DTMC) is used to model systems with probabilistic behavior. Formally, a DTMC $\mathcal{M}$ is a tuple (Z, z_s, Δ) where Z is a countable set of states, $z_s \in Z$ is the initial state and $\Delta : Z \to \mathsf{Dist}(Z)$ is the probabilistic transition function that maps states to a discrete probability distribution over Z. Informally, the process modeled by $\mathcal{M}$ evolves as follows. The process starts in state z_s. After i execution steps, if the process is in state z, the process moves to state z' at execution step $(i+1)$ with probability $\Delta(z)(z')$.

An execution ρ of $\mathcal{M}$ is a sequence of states $z_0 \to z_1 \to z_2 \to \cdots$ such that $z_0 = z_s$ and $z_{i+1} \in \mathsf{support}(\Delta(z_i))$ for all $i \geq 0$. Let $\mathsf{Exec}^{\infty}(\mathcal{M})$ be the set of all executions and $\mathsf{Exec}(\mathcal{M})$ be the set of all finite executions. To each finite execution $\rho_{\mathsf{fin}} = z_0 \to \ldots \to z_m \in \mathsf{Exec}(\mathcal{M})$ we associate a probability $\mathsf{Pr}_{\mathcal{M}}(\rho^{\mathsf{fin}}) = \prod_{i=0}^{m-1} \Delta(z_i)(z_{i+1})$. The cylinder set of ρ_{fin} is $\mathsf{Cyl}(\rho_{\mathsf{fin}}) =$

$\{\rho \in \mathsf{Exec}(\mathcal{M}) | \rho_{\mathsf{fin}} \text{ is a prefix of } \rho\}$. Let S be the smallest σ-algebra on $\mathsf{Exec}^{\infty}(\mathcal{M})$ containing the cylinder sets $\mathsf{Cyl}(\rho_{\mathsf{fin}})$ for all $\rho_{\mathsf{fin}} \in \mathsf{Exec}(\mathcal{M})$. Define $\mathsf{prob}_{\mathcal{M}}$ as the unique probability measure on S such that $\mathsf{prob}_{\mathcal{M}}(\mathsf{Cyl}(\rho_{\mathsf{fin}})) = \mathsf{Pr}_{\mathcal{M}}(\rho_{\mathsf{fin}})$ for all $\rho_{\mathsf{fin}} \in \mathsf{Exec}(\mathcal{M})$.

2.3 MARKOV DECISION PROCESSES

Markov decision processes (MDPs) are used to model systems that exhibit both probabilistic and non-deterministic behavior. An MDP $\mathcal{M}$ is a tuple $(Z, z_s, \mathsf{Act}, \Delta)$ where Z is a countable set of states, $z_s \in Z$ is the initial state, Act is a countable set of actions and $\Delta : Z \times \mathsf{Act} \to \mathsf{Dist}(Z)$ is the probabilistic transition function. $\mathcal{M}$ is said to be finite if the sets Z and Act are finite. An MDP is like a Markov chain except that at each state z, there is a choice among many probabilistic transitions. This choice of which transition to *trigger* is resolved by an *adversary*. Informally, the process modeled by $\mathcal{M}$ evolves as follows. The process starts in state z_s. After i execution steps, if the process is in state z, the attacker chooses an action α such that $\Delta(z, \alpha) = \mu$ and the process moves to state z' in step $(i+1)$ with probability $\mu(z')$.

An execution of an MDP is a sequence $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots$ such that $z_0 = z_s$ and $z_{i+1} \in \mathsf{support}(\Delta(z_i, \alpha_{i+1}))$ for all $i \geq 0$. We will again use $\mathsf{Exec}^{\infty}(\mathcal{M})$ to denote the set of executions and $\mathsf{Exec}(\mathcal{M})$ to denote the set of all finite executions of $\mathcal{M}$. For a finite execution $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} z_2 \cdots \xrightarrow{\alpha_m} z_m$ we write $\mathsf{last}(\rho) = z_m$ and say that ρ has length m, denoted $|\rho| = m$. An execution ρ' is said to be a *one-step extension* of ρ if there exists α_{m+1} and z_{m+1} such that $\rho' = \rho \xrightarrow{\alpha_{m+1}} z_{m+1}$. In such a case, we say ρ' extends ρ by (α_{m+1}, z_{m+1}). An execution is called maximal if it is infinite or if it is finite and has no one-step extension.

An adversary for $\mathcal{M}$ is function $\mathcal{A} : \mathsf{Exec}(\mathcal{M}) \to \mathsf{Dist}(\mathsf{Act})$ that maps finite executions to distributions on actions. An adversary $\mathcal{A}$ for $\mathcal{M}$ resolves all non-determinism and the resulting system can be described by a DTMC $\mathcal{M}^{\mathcal{A}} = (\mathsf{Exec}(\mathcal{M}), z_s, \Delta^{\mathcal{A}})$ where, for each $\rho \in \mathsf{Exec}(\mathcal{M})$, $\Delta^{\mathcal{A}}(\rho)$ is the unique discrete probability distribution that satisfies the following. For each $\rho_1 \in \mathsf{Exec}(\mathcal{M})$, $z \in Z$ and $\alpha \in \mathsf{Act}$, ρ_1 extends ρ by (α, z), $\Delta^{\mathcal{A}}(\rho)(\rho_1) = \mathcal{A}(\rho)(\alpha) \cdot \Delta(\mathsf{last}(\rho), \alpha)(z)$.

Probabilistic Bisimulation. Let $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta)$ be an MDP and $L : Z \to 2^{\mathsf{AP}}$ be a labeling function that maps states to subsets of the set of atomic propositions AP. A probabilistic bisimulation on $\mathcal{M}$ is an equivalence relation $\mathcal{R}$ on Z such that, for all $(z_1, z_2) \in Z$ and $\alpha \in \mathsf{Act}$,

1. $L(z_1) = L(z_2)$

2. $\Delta(z_0, \alpha)(C) = \Delta(z_1, \alpha)(C)$ for all $C \in Z/\mathcal{R}$

where $\Delta(z, \alpha)(C) = \sum_{z' \in C} \Delta(z, \alpha)(z')$. Two MDPs $\mathcal{M}_i = (Z_i, z_s^i, \mathsf{Act}_i, \Delta_i)$ for $i \in \{0, 1\}$ with labeling function L are called *bisimilar*, denoted $\mathcal{M}_0 \sim \mathcal{M}_1$, if there exists a probabilistic bisimulation $\mathcal{R}$ on $\mathcal{M} = \mathcal{M}_0 \uplus \mathcal{M}_1$ such that $(z_s^0, z_s^1) \in \mathcal{R}$.

2.4 PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

Partially observable Markov decision processes (POMDPs) extend MDPs by restricting what an adversary can observe from the states of a system/process. Formally, a POMDP $\mathcal{M}$ is a tuple $(Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where $\mathcal{M}_0 = (Z, z_s, \mathsf{Act}, \Delta)$ is an MDP, $\mathcal{O}$ is a countable set of observations and $\mathsf{obs} : Z \to \mathcal{O}$ is a labeling of states with observations. $\mathcal{M}$ is said to be finite if the sets Z and Act are finite. The set of executions of $\mathcal{M}_0$ is taken to be the set of executions of $\mathcal{M}$, i.e., we define $\mathsf{Exec}(\mathcal{M})$ as the set $\mathsf{Exec}(\mathcal{M}_0)$. We will lift the relevant notation from Section 2.3. Given a finite execution $\rho = z_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} z_n$ of $\mathcal{M}$, the trace of ρ is $\mathsf{tr}(\rho) = \mathsf{obs}(z_0)\alpha_1 \ldots \alpha_n \mathsf{obs}(z_n)$. Let $\mathsf{Trace} = \mathcal{O} \cdot (\mathsf{Act} \cdot \mathcal{O})^*$ be the set that contains $\mathsf{tr}(\rho)$. A finite trace $o_0\alpha_1 \ldots \alpha_n o_n \in \mathsf{Trace}$ is said to have length n, denoted $|\overline{o}| = n$. An adversary for a POMDP is a function $\mathcal{A} : \mathsf{Trace} \to \mathsf{Dist}(\mathsf{Act})$.

Let $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ be a POMDP and $\mathcal{A}$ be an adversary. Recall that $\kappa = \zeta_0 \to \zeta_1 \to \ldots \to \zeta_m \in \mathsf{Exec}(\mathcal{M}^{\mathcal{A}})$ is such that $\zeta_i \in \mathsf{Exec}(\mathcal{M})$ for each $i < m$. For an execution $\rho \in \mathsf{Exec}(\mathcal{M})$, the probability of ρ under adversary $\mathcal{A}$, denoted $\mathsf{prob}_{\mathcal{M}}(\rho, \mathcal{A})$, is $\mathsf{prob}_{\mathcal{M}^{\mathcal{A}}}(\kappa)$ where $\kappa \in \mathsf{Exec}(\mathcal{M}^{\mathcal{A}})$ and $\mathsf{last}(\kappa) = \rho$. For $\kappa \in \mathsf{Exec}(\mathcal{M}^{\mathcal{A}})$ we will write $\mathsf{tr}(\kappa)$ is $\mathsf{tr}(\zeta_m)$ if $\zeta_m \in \mathsf{Exec}(\mathcal{M})$ and $\mathsf{tr}(\zeta_{m-1}) \cdot \zeta_m$ otherwise. For a sequence $\overline{o} \in \mathsf{Trace}$, the probability of $\mathcal{A}$ observing $\overline{o}$, written $\mathsf{prob}_{\mathcal{M}}(\overline{o}, \mathcal{A})$, is the sum of the measures of executions in the set $\{\kappa \in (\mathsf{Exec}(\mathcal{M}^{\mathcal{A}})) \mid \mathsf{tr}(\kappa) = \overline{o}\}$. We may simply write $\mathsf{prob}_{\mathcal{M}}(\overline{o})$ when the adversary is not relevant in a particular context.

A POMDP $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ is said to be acyclic if there is a set of absorbing states $Z_{\mathsf{abs}} \subseteq Z$ such that for all $\alpha \in \mathsf{Act}$ and $z \in Z_a$, $\Delta(z, \alpha)(z) = 1$ and for all $\rho = z_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} \in \mathsf{Exec}(\mathcal{M})$ if $z_i = z_j$ for $i \neq j$ then $z_i \in Z_{\mathsf{abs}}$.

State-based safety properties. Given a POMDP $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$, a set $\psi \subseteq Z$ is said to be a state-based safety property. An execution $\rho = z_0 \xrightarrow{\alpha_1} z_1 \xrightarrow{\alpha_2} \ldots$ is said to satisfy ψ if $z_j \in \psi$ for all $j \geq 0$. An execution $\kappa = \zeta_0 \to \zeta_1 \to \ldots \in \mathsf{Exec}^{\infty}(\mathcal{M}^{\mathcal{A}})$ is said to satisfy ψ, denoted $\kappa \models \psi$, if whenever $\zeta_i \in \mathsf{Exec}^{\infty}(\mathcal{M})$ then ζ_i satisfies ψ. We say that $\mathcal{M}$ satisfies ψ with probability $\geq p$ against adversary $\mathcal{A}$, denoted $\mathcal{M}^{\mathcal{A}} \models_p \psi$, if the sum of the measures of the executions in the set $\{\kappa \in \mathsf{Exec}^{\infty}(\mathcal{M}^{\mathcal{A}}) \mid \kappa \text{ is maximal and } \kappa \models \psi\}$ in

the Markov chain $\mathcal{M}^{\mathcal{A}}$ is $\geq p$. We say that $\mathcal{M}$ satisfies ψ with probability $\geq p$, denoted $\mathcal{M} \models_p \psi$, if for all adversaries $\mathcal{A}$, $\mathcal{M}^{\mathcal{A}} \models_p \psi$.

Indistinguishability. Given two POMDPs $\mathcal{M}_i = (Z_i, z_s^i, \mathsf{Act}_i, \Delta_i, \mathcal{O}_i, \mathsf{obs}_i)$ for $i \in \{0, 1\}$ with the same set of actions and observations, we we say that $\mathcal{M}_0$ and $\mathcal{M}_1$ are distinguishable by an adversary $\mathcal{A}$ if there is an $\overline{o} \in \mathsf{Trace}$ such that $\mathsf{prob}_{\mathcal{M}_0}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{\mathcal{M}_1}(\overline{o}, \mathcal{A})$. If $\mathcal{M}_0$ and $\mathcal{M}_1$ cannot be distinguished by any adversary, they are said to be indistinguishable, denoted $\mathcal{M}_0 \approx \mathcal{M}_1$.

Non-randomized adversaries. An adversary $\mathcal{A} : \mathsf{Trace} \to \mathsf{Dist}(\mathsf{Act})$ is said to be non-randomized if for each $\overline{o} \in \mathsf{Trace}$, $\mathcal{A}(\overline{o})(\alpha)$ is non-zero for exactly one $\alpha \in \mathsf{Act}$. Due to the following proposition, one need only consider non-randomized adversaries when analyzing state-based safety and indistinguishability properties of POMDPs. For the remainder of this work, we will restrict our attention to this class of adversaries.

Proposition 2.1 ([67]) *Let $\mathcal{M}$ and $\mathcal{M}'$ be POMDPs, ψ be a state-based safety property and $\mathcal{A}$ be an adversary. The following are true.*

1. *If $\mathcal{M}^{\mathcal{A}} \not\models_p \psi$ then there exists a non-randomized adversary $\mathcal{A}_0$ such that $\mathcal{M}^{\mathcal{A}_0} \not\models_p \psi$.*
2. *If $\mathcal{M} \not\approx \mathcal{M}'$ then there exists a non-randomized adversary $\mathcal{A}_0$ such that $\mathsf{prob}_{\mathcal{M}}(\overline{o}, \mathcal{A}_0) \neq \mathsf{prob}_{\mathcal{M}'}(\overline{o}, \mathcal{A}_0)$.*

2.5 PROBABILISTIC FINITE AUTOMATA

A PFA is like a finite-state deterministic automaton except that the transition function from a state on a given input is described as a probability distribution. Formally, a PFA A is a tuple $(Q, \Sigma, q_s, \Delta, F)$ where Q is a finite set of states, Σ is a finite input alphabet, $q_s \in Q$ is the initial state, $\Delta : Q \times \Sigma \to \mathsf{Dist}(Q)$ is the transition relation and $F \subseteq Q$ is a set of accepting states. A run ρ of A on an input word $u \in \Sigma^* = a_1 a_2 \ldots a_m$ is a sequence $q_0 q_1 \ldots q_m \in Q^*$ such that $q_0 = q_s$ and $\Delta(q_{i-1}, a_i)(q_i) > 0$ for each $1 \leq i \leq m$. For the run ρ on word u, its measure, denoted $\mathsf{prob}_{\mathsf{A},u}(\rho)$, is $\prod_{i=1}^{m} \Delta(q_{i-1}, a_i)(q_i)$. The run ρ is called *accepting* if $q_m \in F$. The probability of accepting a word $u \in \Sigma$, written $\mathsf{prob}_{\mathsf{A}}(u)$, is the sum of the measures of the accepting runs on u. Two PFAs A_0 and A_1 with the same input alphabet Σ are said to be equivalent, denoted $\mathsf{A}_0 \equiv \mathsf{A}_1$, if $\mathsf{prob}_{\mathsf{A}_0}(u) = \mathsf{prob}_{\mathsf{A}_1}(u)$ for all $u \in \Sigma^*$. Acyclic PFAs are defined analogously to acyclic POMDPs.

2.6 TERMS, EQUATIONAL THEORIES AND FRAMES

A signature $\mathcal{F}$ contains a finite set of function symbols, each with an associated arity and two special countable sets of constant symbols $\mathcal{N}_{\mathsf{pub}}$ and $\mathcal{N}_{\mathsf{priv}}$ representing public and private names, respectively. Variable symbols are the union of two disjoint sets $\mathcal{X}$ and $\mathcal{X}_w$, used to represent protocol and frame variables, respectively. The sets $\mathcal{F}$, $\mathcal{N}_{\mathsf{pub}}$, $\mathcal{N}_{\mathsf{priv}}$, $\mathcal{X}$ and $\mathcal{X}_w$ are required to be pairwise disjoint. Terms are built by the application of function symbols to variables and terms in the standard way. Given a signature $\mathcal{F}$ and $\mathcal{Y} \subseteq \mathcal{X} \cup \mathcal{X}_w$, we use $\mathcal{T}(\mathcal{F}, \mathcal{Y})$ to denote the set of terms built over $\mathcal{F}$ and $\mathcal{Y}$. The set of variables occurring in a term u is denoted by $\mathsf{vars}(u)$. A ground term is one that contains no free variables.

A substitution σ is a partial function with a finite domain that maps variables to terms, where $\mathsf{dom}(\sigma)$ will denote the domain and $\mathsf{ran}(\sigma)$ will denote the range. For a substitution σ with $\mathsf{dom}(\sigma) = \{x_1, \ldots, x_k\}$, we will denote σ as $\{x_1 \mapsto \sigma(x_1), \ldots, x_k \mapsto \sigma(x_k)\}$. A substitution σ is said to be ground if every term in $\mathsf{ran}(\sigma)$ is ground and a substitution with an empty domain will be denoted as $\emptyset$. Substitutions can be extended to terms in the usual way and we write $t\sigma$ for the term obtained by applying the substitution σ to the term t.

Our process algebra is parameterized by an equational theory $(\mathcal{F}, E)$, where $\mathcal{F}$ is a signature and E is a set of $\mathcal{F}$-Equations. By an $\mathcal{F}$-Equation, we mean a pair $u = v$ where $u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathcal{X})$ are terms that do not contain private names.

Example 2.1 *We can model primitives for symmetric encryption/decryption and a hash function using the equational theory* $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ *with signature* $\mathcal{F}_{\mathsf{senc}} = \{\mathsf{senc}/2,\ \mathsf{sdec}/2,\ \mathsf{h}/1\}$ *and equations* $E_{\mathsf{senc}} = \{\mathsf{sdec}(\mathsf{senc}(m, k), k) = m\}$.

Two terms u and v are said to be equal with respect to an equational theory $(\mathcal{F}, E)$, denoted $u =_E v$, if $E \vdash u = v$ in the first order theory of equality. For equational theories defined in the preceding manner, if two terms containing names are equivalent, they will remain equivalent when the names are replaced by arbitrary terms. We often identify an equational theory $(\mathcal{F}, E)$ by E when the signature is clear from the context. An equational theory E is said to be trivial if $u =_E v$ for any terms u and v and otherwise it is said to be non-trivial. For the remainder of this work, we will assume equational theories are non-trivial. Processes are executed in an environment that consists of a frame $\varphi : \mathcal{X}_w \to \mathcal{T}(\mathcal{F})$ and a binding substitution $\sigma : \mathcal{X} \to \mathcal{T}(\mathcal{F})$.

Definition 2.1 *Two frames* φ_1 *and* φ_2 *are said to be statically equivalent in equational theory* E*, denoted* $\varphi_1 \equiv_E \varphi_2$*, if* $\mathsf{dom}(\varphi_1) = \mathsf{dom}(\varphi_2)$ *and for all* $r_1, r_2 \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathcal{X}_w)$ *we have* $r_1\varphi_1 =_E r_2\varphi_1$ *iff* $r_1\varphi_2 =_E r_2\varphi_2$.

Intuitively, two frames are statically equivalent if an attacker cannot distinguish between the information they contain. A term $u \in \mathcal{T}(\mathcal{F})$ is deducible from a frame φ with recipe $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathsf{dom}(\varphi))$ in equational theory E, denoted $\varphi \vdash^r_E u$, if $r\varphi =_E u$. We often omit r and E and write $\varphi \vdash u$ if they are clear from the context.

A term u can be represented as a labeled directed acyclic graph (DAG), denoted $\mathsf{dag}(u)$ [68, 69]. Every node in $\mathsf{dag}(u)$ is labeled by a function symbol, name or a variable. Nodes labeled by names and variables have out-degree 0. A node labeled with a function symbol f has out-degree equal to the arity of f where outgoing edges of the node are labeled from 1 to the arity of f. Every node of $\mathsf{dag}(u)$ represents a unique sub-term of u. The dag-size of a term u is the number of nodes in $\mathsf{dag}(u)$. We assume that the size of a term u, denoted $|u|$, is the dag-size of u. The depth of a term u, denoted $\mathsf{depth}(u)$, is the length of the longest path in $\mathsf{dag}(u)$.

2.7 PROCESS SYNTAX

We assume a countably infinite set of labels $\mathcal{L}$ and an equivalence relation $\sim$ on $\mathcal{L}$ that induces a countably infinite set of equivalence classes. For $\ell \in \mathcal{L}$, $[\ell]$ denotes the equivalence class of ℓ. We use $\mathcal{L}_b$ and $\mathcal{L}_c$ to range over subsets of $\mathcal{L}$ such that $\mathcal{L}_b \cap \mathcal{L}_c = \emptyset$ and both $\mathcal{L}_b$ and $\mathcal{L}_c$ are closed under $\sim$. Each equivalence class is assumed to contain a countably infinite set of labels. Operators in our grammar will come with a unique label from $\mathcal{L}$, which together with the relation $\sim$, will be used to mask the information an attacker can obtain about actions of a process. When an action with label ℓ is executed, the attacker will only be able to infer $[\ell]$.

Processes in our calculus are the parallel composition of roles, which intuitively are used to model a single actor in a system/protocol. Roles, in turn, are constructed by combining atomic actions through sequential composition and probabilistic choice. Formally, an atomic action is derived from the grammar

$$A \;:=\; 0 \,\Big|\, \nu x^\ell \,\Big|\, (x := u)^\ell \,\Big|\, [c_1 \wedge \ldots \wedge c_k]^\ell \,\Big|\, \mathtt{in}(x)^\ell \,\Big|\, \mathtt{out}(u)^\ell$$

where $\ell \in \mathcal{L}$, $x \in \mathcal{X}$ and $c_i \in \{\top, u = v\}$ for all $i \in \{1, \ldots, k\}$ where $u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathcal{X})$. In the case of the assignment rule $(x := u)^\ell$, we additionally require that $x \notin \mathsf{vars}(u)$. A role is derived from the grammar

$$R \;:=\; A \,\Big|\, (R \cdot R) \,\Big|\, (R +^\ell_p R)$$

where $p \in [0, 1]$, $\ell \in \mathcal{L}$ and $x \in \mathcal{X}$. The 0 process does nothing. The process νx^ℓ creates a fresh name and binds it to x while $(x := u)^\ell$ assigns the term u to the variable x. The test

process $[c_1 \wedge \ldots \wedge c_k]^\ell$ terminates if c_i is $\top$ or c_i is $u = v$ where $u =_E v$ for all $i \in \{1, \ldots, k\}$ and otherwise, if some c_i is $u = v$ and $u \neq_E v$, the process deadlocks. The process $\mathtt{in}(x)^\ell$ reads a term u from the public channel and binds it to x and the process $\mathtt{out}(u)^\ell$ outputs a term on the public channel. The processes $R \cdot R'$ sequentially executes R followed by R' whereas the process $R +_p^\ell R'$ behaves like R with probability p and like R' with probability $1 - p$. Note that protocols in our formalism are *simple*; a protocol is said to be simple if there is no principal-level nondeterminism [70].

We will use P and Q to denote processes, which are the parallel composition of a finite set of roles $R_1, \ldots, R_n$, denoted $R_1 \mid \ldots \mid R_n$. For a process Q, $\mathsf{fv}(Q)$ and $\mathsf{bv}(Q)$ denote the set of variables that have some free or bound occurrence in Q, respectively. The formal definition is standard and is presented in Appendix A. Processes containing no free variables are called ground. We restrict our attention to processes that do not contain variables with both free and bound occurrences. That is, for a process Q, $\mathsf{fv}(Q) \cap \mathsf{bv}(Q) = \emptyset$.

Definition 2.2 *A process $Q = R_1 \mid \ldots \mid R_n$ is said to be well-formed if the following hold.*

1. *Every atomic action and probabilistic choice in Q has a unique label.*
2. *If label ℓ_1 (resp. ℓ_2) occurs in the role R_i (resp. R_j) for $i, j \in \{1, \ldots, n\}$ then $i \neq j$ iff $[\ell_1] \neq [\ell_2]$.*

For the remainder of this work, processes are assumed to be well-formed. Unless otherwise stated, will will also assume that the labels occurring a role come from the same equivalence class.

Convention 2.1 *For readability, we will omit process labels when they are not relevant in a particular context.*

We now give some examples illustrating the type of protocols that can be modeled in our process algebra.

Example 2.2 *We model the electronic voting protocol from Example 1.1 in our formalism. The protocol is built over the equational theory $(\mathcal{F}_{\mathsf{aenc}}, E_{\mathsf{aenc}})$ with signature*

$$\mathcal{F}_{\mathsf{aenc}} = \{\mathsf{sk}/1,\ \mathsf{pk}/1,\ \mathsf{aenc}/3,\ \mathsf{adec}/2,\ \mathsf{fst}/1,\ \mathsf{snd}/1,\ \mathsf{pair}/2\}$$

and equations

$$E_{\mathsf{aenc}} = \{\mathsf{adec}(\mathsf{aenc}(m, r, \mathsf{pk}(k)), \mathsf{sk}(k)) = m,\ \mathsf{fst}(\mathsf{pair}(x_1, x_2)) = x_1,\ \mathsf{snd}(\mathsf{pair}(x_1, x_2)) = x_2\}.$$

For generation of their pubic key pairs; Alice, Bob and the election authority hold private names k_A, k_B and k_{EA}, respectively. The candidates will be modeled using public names c_0 and c_1 and the tokens will be modeled using private names t_A and t_B. Additionally, we will write y_i and r_i for $i \in \mathbb{N}$ to denote fresh input variables and private names, respectively. The roles of Alice, Bob and the election authority are as follows.

$$\begin{array}{lcl} A(c_A) & := & \mathtt{in}(y_0) \cdot \mathtt{out}(\mathsf{aenc}(\mathsf{pair}(\mathsf{adec}(y_0, \mathsf{sk}(k_A)), c_A), r_0, \mathsf{pk}(k_{EA}))) \\ B(c_B) & := & \mathtt{in}(y_1) \cdot \mathtt{out}(\mathsf{aenc}(\mathsf{pair}(\mathsf{adec}(y_1, \mathsf{sk}(k_B)), c_B), r_1, \mathsf{pk}(k_{EA}))) \\ EA & := & \mathtt{out}(\mathsf{aenc}(t_A, r_2, \mathsf{pk}(k_A))) \cdot \mathtt{out}(\mathsf{aenc}(t_B, r_3, \mathsf{pk}(k_B))) \cdot \mathtt{in}(y_3) \cdot \mathtt{in}(y_4) \cdot \\ & & [\mathsf{fst}(\mathsf{adec}(y_3, sk(k_{EA}))) = t_A \wedge \mathsf{fst}(\mathsf{adec}(y_4, sk(k_{EA}))) = t_B] \cdot \\ & & \mathtt{out}(\mathsf{snd}(\mathsf{adec}(y_3, \mathsf{sk}(k_{EA}))) +_{\frac{1}{2}} \mathsf{snd}(\mathsf{adec}(y_4, \mathsf{sk}(k_{EA})))) \end{array}$$

The entire protocol is $\mathsf{evote}(c_A, c_B) = A(c_A) \mid B(c_B) \mid EA$.

For some examples, it is necessary permute the order in which a set of atomic actions $a_1, \ldots, a_n$ are executed. We introduce the shorthand $\mathsf{perm}[a_1, \ldots, a_n]$ for such an operation, which is defined inductively using the basic probabilistic choice operator as follows. If $n = 2$, then $\mathsf{perm}[a_1, a_2] = (a_1 \cdot a_2) +_{\frac{1}{2}} (a_2 \cdot a_1)$. Inductively, if $n > 2$, then $\mathsf{perm}[a_1, \ldots, a_n]$ is

$$P_1 +_{\frac{1}{n}} (P_2 +_{\frac{1}{n-1}} (P_3 +_{\frac{1}{n-2}} \ldots (P_{n-1} +_{\frac{1}{2}} P_n) \ldots)$$

where $P_i = a_i \cdot \mathsf{perm}[a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n]$.

Example 2.3 *A mix-network, originally introduced by Chaum in [9], is a routing protocol used to hide the origin of messages that pass through it. This is achieved by routing messages through a series of proxy servers, called* mixes*, which receive encrypted traffic from multiple senders, shuffle the messages and forward them in random order (for more details, see Section 3.1). More formally, assume there are users $A_1, \ldots, A_n$ who want to communicate anonymously through a single mix server M with users $B_1, \ldots, B_n$, respectively. The protocol is build over the equational theory $(\mathcal{F}_{\mathsf{aenc}}, E_{\mathsf{aenc}})$ from Example 2.2. For generation of their pubic key pairs, the parties $A_0, \ldots, A_n$ (resp. $B_1, \ldots, B_n$), will hold private names $k_{A_1}, \ldots, k_{A_n}$ (resp. $k_{B_1}, \ldots, k_{B_n}$). The mix will hold the private name k_M. We will also assume a set of private names $r_1, r_2, \ldots$ to model nonces and a set of private names $m_1, m_2, \ldots$ to model messages. The behavior of user A_i (for $i \in \{1, \ldots, n\}$) and the mix can be described by the roles below.*

$$\begin{array}{lcl} A_i & := & \mathtt{out}(\mathsf{aenc}(\mathsf{aenc}(m_i, r_i, \mathsf{pk}(k_{B_i})), r_i', \mathsf{pk}(k_M))) \\ M & := & \mathtt{in}(z_1) \cdot \ldots \cdot \mathtt{in}(z_n) \cdot \\ & & \mathsf{perm}(\mathsf{adec}(z_1, \mathsf{sk}(k_M)), \ldots, \mathsf{adec}(z_n, \mathsf{sk}(k_M))) \end{array}$$

2.8 PROCESS SEMANTICS

Given a process P, an extended process is a 3-tuple (P, φ, σ) where φ is a frame and σ is a binding substitution. Semantically, a ground process P over equational theory $(\mathcal{F}, E)$ is a POMDP $[\![P]\!] = (Z \cup \{\mathsf{error}\}, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where Z is the set of all extended processes $z_s = (P, \emptyset, \emptyset)$, $\mathsf{Act} = (\mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathcal{X}_w) \cup \tau) \times \mathcal{L}/\sim$ and $\Delta, \mathcal{O}, \mathsf{obs}$ are defined below. Let $\mu \cdot Q$ denote the distribution μ_1 such that $\mu_1(P', \varphi, \sigma) = \mu(P, \varphi, \sigma)$ if P' is $P \cdot Q$ and 0 otherwise. The distributions $\mu \mid Q$ and $Q \mid \mu$ are defined analogously. For a conjunct c_i ($i \in \{1, \ldots, n\}$) in a test process $[c_1 \wedge \ldots \wedge c_n]$ and a substitution σ we write $c_i \vdash \top$ when c_i is $\top$ or c_i is $u = v$ where $\mathsf{vars}(u, v) \subseteq \mathsf{dom}(\sigma)$ and $u\sigma =_E v\sigma$. We define Δ in Figure 2.1, where we write $(P, \varphi, \sigma) \xrightarrow{\alpha} \mu$ if $\Delta((P, \varphi, \sigma), \alpha) = \mu$. For any extended process (P, φ, σ) and action $\alpha \in \mathsf{Act}$, if $\Delta((P, \varphi, \sigma), \alpha)$ is undefined in Figure 2.1 then $\Delta((P, \varphi, \sigma), \alpha) = \delta_{\mathsf{error}}$. Note that Δ is well-defined, as roles are deterministic and each equivalence class on labels identifies at most one role. For a frame φ and equational theory E, we write $[\varphi]$ to denote the equivalence class of φ with respect to the static equivalence relation $\equiv_E$. We use EQ to denote the set of all such equivalence classes. Let $\mathcal{O} = \mathsf{EQ}$ and define obs as a function from extended processes to $\mathcal{O}$ such that for any extended process $\eta = (P, \varphi, \sigma)$, $\mathsf{obs}(\eta) = [\varphi]$. Given an action α, $\mathsf{depth}(\alpha) = 0$ if $\alpha = (\tau, j)$ and $\mathsf{depth}(\alpha) = m$ if $\alpha = (r, j)$ and $\mathsf{depth}(r) = m$. For a frame/substitution σ, $|\sigma|$ is the sum of the sizes of the terms in $\mathsf{ran}(\sigma)$.

Figure 2.1 *Process semantics.*

$$\frac{r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathcal{X}_w) \quad \varphi \vdash^r u \quad x \notin \mathsf{dom}(\sigma)}{(\mathtt{in}(x)^\ell, \varphi, \sigma) \xrightarrow{(r,[\ell])} \delta_{(0,\varphi,\sigma \cup \{x \mapsto u\})}} \text{ IN}$$

$$\frac{x \notin \mathsf{dom}(\sigma) \quad n \text{ is a fresh name}}{(\nu x^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi,\sigma \cup \{x \mapsto n\})}} \text{ NEW}$$

$$\frac{\mathsf{vars}(u) \subseteq \mathsf{dom}(\sigma) \quad i = |\mathsf{dom}(\varphi)| + 1}{(\mathtt{out}(u)^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi \cup \{w_{(i,[\ell])} \mapsto u\sigma\}, \sigma)}} \text{ OUT}$$

$$\frac{Q_0 \neq 0 \quad (Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \cdot Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu \cdot Q_1} \text{ SEQ}$$

$$\frac{\forall i \in \{1, \ldots, n\}, c_i \vdash \top}{([c_1 \wedge \ldots \wedge c_n]^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi,\sigma)}} \text{ TEST}$$

$$\frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(0 \cdot Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu} \text{ NULL}$$

$$\frac{\mathsf{vars}(u) \subseteq \mathsf{dom}(\sigma) \quad x \notin \mathsf{dom}(\sigma)}{((x := u)^\ell, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(0,\varphi,\sigma \cup \{x \mapsto u\sigma\})}} \text{ ASGN}$$

$$\frac{(Q_0, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \mid Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu \mid Q_1} \text{ PAR}_{\text{L}}$$

$$\frac{}{(Q_1 +_p^\ell Q_2, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(Q_1,\varphi,\sigma)} +_p \delta_{(Q_2,\varphi,\sigma)}} \text{ PROB}$$

$$\frac{((Q_1, \varphi, \sigma) \xrightarrow{\alpha} \mu}{(Q_0 \mid Q_1, \varphi, \sigma) \xrightarrow{\alpha} Q_0 \mid \mu} \text{ PAR}_{\text{R}}$$

Protocols P and P' are said to indistinguishable if $[\![P]\!] \approx [\![P']\!]$. As observed in [67], this no-

tion of indistinguishability coincides with the classical notion of trace-equivalence for simple non-randomized protocols. In Examples 2.4 and 2.5 below, we demonstrate how properties of randomized security protocols can be specified using the notion of indistinguishability.

Example 2.4 *Consider the simple electronic voting protocol from Example 2.2. We say that the protocol satisfies the vote privacy property if* $\mathsf{evote}(c_0, c_1)$ *and* $\mathsf{evote}(c_1, c_0)$ *are indistinguishable.*

Example 2.5 *Let* $P = A_1 \mid \ldots \mid A_n \mid M$ *be the mix network protocol from Example 2.3. Let*

$$\mathcal{M}' = \mathtt{in}(z_1) \cdot \ldots \cdot \mathtt{in}(z_n) \cdot \mathsf{perm}(\mathsf{aenc}(z_1, r_1, \mathsf{pk}(k_{B_1})), \ldots, \mathsf{aenc}(z_n, r_n, \mathsf{pk}(k_{B_1})))$$

be an idealized mix that outputs the messages from $A_1, \ldots, A_n$ *in random order. The protocol* P *preserves the anonymity of senders if* P *is indistinguishable from* $A_1 \mid \ldots \mid A_n \mid M'$*.*

Definition 2.3 *An extended process* (P, φ, σ) *preserves the secrecy of a term* u *in the equational theory* $(\mathcal{F}, E)$*, denoted* $(P, \varphi, \sigma) \models_E u$*, if there is no* $r \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathsf{dom}(\varphi))$ *such that* $\varphi \vdash^r_E u\sigma$*. We write* $\mathsf{secret}(u)$*, to represent the set of states of* $[\![P]\!]$ *that preserve the secrecy of* u *and* $\mathsf{secret}(\{u_1, \ldots, u_n\})$ *to denote* $\mathsf{secret}(u_1) \cap \ldots \cap \mathsf{secret}(u_n)$*.*

Notation 2.1 *For a process* P *and terms* $u_1, \ldots, u_n$*,* $\mathsf{secret}(\{u_1, \ldots, u_n\})$ *is a state-based safety property of* $[\![P]\!]$*. For a probability* p*, we will write* $P \models_{E,p} \mathsf{secret}(u_1, \ldots, u_n)$*, if* $[\![P]\!] \models_p \mathsf{secret}(\{u_1, \ldots, u_n\})$*.*

In the following examples, we demonstrate how the security properties from Examples 2.4 and 2.5 can be reformulated as state-base safety properties.

Example 2.6 *Consider the simple electronic voting protocol from Example 2.2. We will alter the protocol such that vote privacy can be modeled as a state-based safety property. In our modeling, each of the voters will toss a fair coin to decide among the candidates* c_0 *and* c_1*. After the election authority publishes the votes, the adversary will then be given a chance to "guess" which candidate was chosen by Alice. If the guess is correct, a special public name* s *will be output. Formally, the roles of Alice, Bob and the special test process* S *are given below.*

$$\begin{aligned}
A' &:= \mathtt{in}(y_0) \cdot ((c_A := c_0) +_{\frac{1}{2}} (c_A := c_1)) \cdot \\
&\quad\ \mathtt{out}(\mathsf{aenc}(\mathsf{pair}(\mathsf{adec}(y_0, \mathsf{sk}(k_A)), c_A), r_0, \mathsf{pk}(k_{EA}))) \\
B' &:= \mathtt{in}(y_1) \cdot ((c_B := c_0) +_{\frac{1}{2}} (c_B := c_1)) \cdot \\
&\quad\ \mathtt{out}(\mathsf{aenc}(\mathsf{pair}(\mathsf{adec}(y_1, \mathsf{sk}(k_B)), c_B), r_1, \mathsf{pk}(k_{EA}))) \\
S &:= \mathtt{in}(y_s) \cdot [y_s = c_A] \cdot \mathtt{out}(s)
\end{aligned}$$

Vote privacy can be modeled as through the secrecy of the secret s. Notice that if both Alice and Bob chose the same candidate (which occurs with probability $\frac{1}{2}$) then the attacker knows Alice's vote. If Alice and Bob chose different candidates (which also occurs with probability $\frac{1}{2}$) then the random permutation of the votes performed by the election authority should not allow the attacker to distinguish the scenario in which Alice votes for c_0 and Bob votes for c_1 from the one in which Alice votes for c_1 and Bob votes for c_0. That is, the protocol satisfies vote privacy if s is kept secret with probability $\frac{1}{4}$, i.e. $(A' \mid B' \mid EA \mid S) \models_{E_{\mathsf{aenc}}, \frac{1}{4}} \mathsf{secret}(s)$.

Example 2.7 *Consider the mix-net protocol $P = A_1 \mid \ldots \mid A_n \mid M$ defined in Example 2.3. The protocol is designed to ensure that the messages output by the mix cannot be linked to the original sends with high probability. That is, the adversary should be able to do no better than "guess" which output message belongs to which sender. This hypothesis is violated if, for a mix output, the adversary can identify the sender of the message with probability $> \frac{1}{n}$. We can model this property in our framework by adding, for each $i \in \{1, \ldots, n\}$, a role*

$$S_i = \mathtt{in}(z_i') \cdot [z_i' = \mathsf{aenc}(m_i, n_i, \mathsf{pk}(k_{B_i}))] \cdot \mathtt{out}(s_i)$$

to the process, where s_i is a private name. The protocol P preserves the anonymity of sender A_i if $(A_0 \mid \ldots \mid A_n \mid M \mid S_1 \mid \ldots \mid S_n) \models_{E_{\mathsf{aenc}}, \frac{1}{n}} \mathsf{secret}(s_i)$.

CHAPTER 3: RANDOMIZED SECURITY PROTOCOLS

In this chapter, we describe a group of randomized security protocols that will serve as running examples throughout the remainder of this work. This group of protocols also constitutes the benchmark suite upon which we evaluate our protocol analysis tool (Chapter 4).

3.1 MIX-NETWORKS

A mix-network [9], is a routing protocol used to break the link between a message's sender and receiver. This is achieved by routing messages through a series of proxy servers, called mixes. Each mix collects a batch of encrypted messages, privately decrypts each message and forwards the resulting messages in random order. More formally, consider a sender Alice (A) who wishes to send a message m to Bob (B) through Mix (M). Alice prepares a cipher-text of the form

$$\mathsf{aenc}(\mathsf{aenc}(m, n_1, \mathsf{pk}(B)), n_0, \mathsf{pk}(M))$$

where aenc is asymmetric encryption, n_0, n_1 are nonces and $\mathsf{pk}(M)$, $\mathsf{pk}(B)$ are the public keys of the Mix and Bob, respectively. Upon receiving a batch of N such cipher-texts, the Mix unwraps the outer layer of encryption on each message using its secret key and then randomly permutes and forwards the messages. This step of the protocol is referred to as a *flush*. A passive attacker, who observes all traffic but does not otherwise modify messages on the network, cannot (with high probability) correlate messages entering and exiting the Mix. Unfortunately, this simple design, known as a *threshold mix*, is vulnerable to an active attack. To expose Alice as the sender of the message $\mathsf{aenc}(m, n_1, \mathsf{pk}(B))$, an attacker simply forwards Alice's message along with $N-1$ dummy messages to the Mix. In this way, the attacker can distinguish which of the Mix's N output messages is not a dummy message and hence must have originated from Alice. Although active attacks of this nature cannot be thwarted completely, several mix-network designs have been proposed to increase the overhead associated with carrying out such an attack. We describe several of these designs below. Each design will deviate from the mix described above only in its algorithm for flushing messages.

Pool mixes. In each round, a *pool mix* [71, 72, 73], receives a set of messages, adds them to its *pool*, and outputs a random subset of the messages from its pool in random order. The remainder of the message are retained as part of the pool for later rounds. Often times,

when the mix is first instantiated, the pool will be initialized with a set of dummy messages that are indistinguishable from actual network traffic. Various methodologies regarding which subsets of messages should be flushed or retained in each round has lead to several variations of the pool mix design.

In the simplest instantiation of a pool mix, a constant number of messages m are retained in the pool at each round. After a flush, the mix collects an additional n messages. Once $n+m$ messages are held, the mix randomly selects another n messages to flush. This design is known as a *threshold pool mix*. As opposed to retaining a constant number of messages in the pool at each round, a *dynamic pool mix* outputs only a fraction f (for $0 < f \leq 1$) of the new messages it receives at each round. If the mix holds $n + m$ messages at the end of a round, then $\max(1, \lfloor n \cdot f \rfloor)$ of the messages are forward.

Pool mixes do not eliminate the threat of $N-1$ flooding attacks. However, they impose a much larger burden on an adversary attempting to carry out such an attack. In the case of threshold mixes, a single round is sufficient to positively link a message with its sender. In the case of pool mixes, the $N-1$ flooding attack becomes probabilistic in nature. A message is retained in the pool for an unpredictable number of rounds. But, as the number of rounds increases, the probability of a message remaining in the pool decreases. In [74], a study of the effectiveness of $N-1$ flooding attacks details the number of rounds required to attack each of the pool mix variations described above.

Binomial mixes. A *binomial mix* [75, 76, 77] adopts a dynamic strategy for its flushing algorithm. For each $n \in \mathbb{N}$, the mix has an associated probability of forwarding, denoted $p_f(n)$. At the end of a round, if the mix holds n messages, a biased coin is flipped for each message. With probability $p_f(n)$ a message is forwarded and with probability $1 - p_f(n)$ a message is retained for later rounds. In this approach, the number of messages output each round is selected from a binomial distribution with mean $n \cdot p_f(n)$ and variance $n \cdot p_f(n)(1 - p_f(n))$.

Mix network topology. Observe that, in all of the mix-network schemes, the mix server constitutes a single point of trust (failure). If the mix does not faithfully execute the protocol or correctly randomize its outputs, than the security of the entire scheme is broken. As a result, mix-networks are typically deployed using multiple mix servers. Incoming messages are routed through a subset of these servers according to a statically or dynamically selected path. In this way, trust is distributed among several of the servers in the mix-network.

In the *cascade mix* design, a user selects a static path for his/her message through the mix network. This is achieved by a re-encrypting a message several times, each with the public

key of a different mix server. Upon receiving a cipher-text, each mix in the encryption chain unwraps one layer of the encryption and forwards the resulting cipher-text to a new mix in the chain. An alternative approach, called a *free-route mix*, allows the route of a message to be dynamically selected by the mix-network. A sender performs a single encryption of his/her message and forwards it to a mix server of its choice. This mix then unwraps the outer layer of encryption, re-encrypts the message under the public key of another mix in the network and forwards the message to that mix. Eventually one of the mix servers in this communication chain forwards the message to the intended recipient. Each of the network topologies described above comes with its own set of advantages and drawbacks. There has been some debate in the research community as to which design provides a higher degree of security [78, 79].

3.2 DINNING CRYPTOGRAPHERS NETWORKS

In a simple DC-net protocol [80, 41], two parties Alice and Bob want to anonymously publish two confidential bits m_A and m_B, respectively. To achieve this, Alice and Bob agree on three private random bits b_0, b_1 and b_2 and output a pair of messages according to the following scheme.

$$\begin{array}{lll} \text{If } b_0 = 0 & \text{Alice:} & M_{A,0} = b_1 \oplus m_A,\ M_{A,1} = b_2 \\ & \text{Bob:} & M_{B,0} = b_1,\ M_{B,1} = b_2 \oplus m_B \\ \text{If } b_0 = 1 & \text{Alice:} & M_{A,0} = b_1,\ M_{A,1} = b_2 \oplus m_A \\ & \text{Bob:} & M_{B,0} = b_1 \oplus m_B,\ M_{B,1} = b_2 \end{array}$$

From the protocol output, the messages m_A and m_B can be retrieved as $M_{A,0} \oplus M_{B,0}$ and $M_{A,1} \oplus M_{B,1}$. The party to which the messages belong, however, remains unconditionally private, provided the exchanged secrets are not revealed.

The protocol can be extended to multiple parties as follows. Let $A_1, \ldots, A_n$ be the participants for the protocol. Each pair of participants (A_i, A_j) shares a set of secret keys $k_{i,j}(w)$ for $i, j, w \in \{1, \ldots, n\}$ where $k_{i,j}(w) = k_{j,i}(w)$. To broadcast the messages of parties participating in the scheme, each member A_i first computes a vector of values

$$W_i = [W_i(1) = \oplus_{j=1}^{n} k_{i,j}(1), \ldots, W_i(n) = \oplus_{j=1}^{n} k_{i,j}(n)]$$

where each $W_i(w)$ is called a *pad*. A_i then chooses a random position c_i and and computes a new vector

$$V_i = [W_i(1), \ldots, W_i(c_i) \oplus m_i, \ldots, W_i(n)]$$

where m_i is the message of A_i. Each party then outputs the message V_i from which the set of all messages can be obtained as $V = \oplus_{i=1}^{n}(V_i)$ as long each party selected a distinct position c_i.

3.3 3-BALLOT ELECTRONIC VOTING

In this section, we present the 3-ballot voting system from [17]. To simplify the presentation of the protocol, we begin by describing the major concepts behind 3-ballot voting schemes, as originally introduced in [42]. At the polling station, each voter is given 3 ballots at random. A ballot is comprised of a list of candidates and a ballot ID. When casting a vote, a voter begins by placing exactly one mark next to each candidate on one of the three ballots, chosen a random. An additional mark is then placed next to the desired candidate on one of the ballots, again chosen at random. At the completion of the procedure, at least one mark should have been placed on each ballot and two ballots should have marks corresponding to the desired candidate. Ballots are then collected by an election authority, however, each voter retains a copy of one of the ballots as a receipt. Once all votes have been collected, the election authority publishes all of the ballots after which each voter can use his/her receipt to verify their vote was cast. The full protocol from [17] is as follows. The protocol will consist of three phases. There will be a set of voters, a registration agent (R) and a voting manager (M).

In the first phase, called the *registration phase*, R begins by creating a voters/ballot ID repository (VBR), requesting b ballot IDs from M and storing them in the VBR. The VBR contains information the registration agent needs to verify the eligibility of a voter. When a voter enters the polling station, R verifies his/her right to vote and then takes three ballot IDs from the VBR an signs them with his/her private key. The signed ballot IDs are then given to the voter after which R queries M for three replacement ballot IDs. Note that all of the ballot IDs are encrypted under the public key of a voting console, which is a physical device used to collect votes. In particular, this means that R does not know the value of the ballot IDs.

The second phase of the protocol is the *voting phase*. Here a voter proceeds with the three signed ballot IDs it obtained in the previous phase to a voting console. The voting console uses verifies the signature of R on each ballot ID through the public key infrastructure (PKI) and decrypts the ballot IDs. The first ballot ID is chosen as the receipt ballot ID (RID) and sent to M. All of the communication between M and the voting console is done over an encrypted and authenticated channel. M verifies that the RID has not been used before, to prevent a replay attack. The voting manager then sends three ballots to the voting console.

The voter marks the three ballots according to the scheme presented at the beginning of this section. After voting, the voter randomly selects one of the ballots to maintain as a receipt. The chosen ballot is assigned the RID and the other two ballots are given the remaining ballot IDs. The voting console then encrypts each of the ballots and sends them to M in random order. Each of the three ballots is encrypted under the public key of one of three vote repositories. Once the voting manager receives the three ballots, he/she signs them and forwards them to the appropriate vote repository. All of the vote repositories validate the signature of M before storing the vote in a random position.

Once all vote have been cast, the *vote storage and counting phase* commences. In this final phase, a vote counting mechanism queries all of the vote repositories. Upon receiving the votes, the vote counter posts the votes along with the ballot IDs to a bulletin board. Voters can verify their vote was counted by ensuring the ballot ID they retained as a receipt appears on the bulletin board.

3.4 FUJIOKA, OKAMOTO AND OHTA ELECTRONIC VOTING

The Fujioka, Okamoto and Ohta (FOO) voting protocol [43], is comprised of 3 phases carried out by a set of voters $(V_1, \ldots, V_n)$, an administrator (A) and a collector (C). To ensure the privacy of voters from the administrator, both bit commitment [81] and blind signature [82] schemes are used. For such schemes, will write $\xi(v, k)$ to denote the commitment to message v using key k and $\chi(m, b)$ for the blinding of message m with blinding factor b. By $\sigma_i(m)$ (resp. $\sigma_A(m)$), we mean the digital signature scheme of voter V_i (resp. administrator A). The full protocol is below.

Phase 1:

- V_i commits to vote v_i as $x_i = \xi(v_i, k_i)$
- V_i computes $e_i = \chi(x_i, b_i)$ and sends the pair $\langle V_i, \sigma_i(e_i)\rangle$ to A
- A verifies that V_i has the right to vote, has not yet voted, and the signature $\sigma_i(e_i)$ is valid
- If the preceding hold, A sends $\sigma_A(e_i)$ to V_i
- V_i unblinds $\sigma_A(e_i)$ to obtain $y_i = \sigma_A(x_i)$

Phase 2:

- V_i sends y_i to C over an anonymous channel
- C verifies the signature of y_i and stores the tuple $\langle \ell_i, x_i, y_i \rangle$ for number ℓ_i

Phase 3:

- After collecting all votes, C publishes every $\langle \ell_i, x_i, y_i \rangle$
- V_i verifies $\langle \ell_i, x_i, y_i \rangle$ was published and sends $\langle \ell_i, k_i \rangle$ to C over an anonymous channel
- C opens x_i using k_i to retrieve v_i and validates it is a well-formed vote
- Once all votes are collected the election results are released

The protocol assumes the existence of a perfectly anonymous channel. Unfortunately, such a channel is difficult to realize in practice and, to the best of our knowledge, no implementations have been proposed in the literature. In fact, our analysis of the protocol led us to conclude that such channels cannot not be realized without modification to the protocol itself. To facilitate a concrete analysis, we a propose one such modification that instantiates perfectly anonymous channels using mix-networks. The required changes to the protocol are subtle, and many natural channel implementations can lead to a degradation in security. Below we give our proposed modifications and enhancements to phases 2 and 3 of the FOO protocol (phase 1 remains unchanged). We will write n_i, r_i and r'_i to denote nonces held by voter V_i and $\mathsf{pk}(m)$ to denote the public key of the mix.

Phase 2*:

- V_i sends $\mathsf{aenc}(\langle y_i, n_i \rangle, r_i, \mathsf{pk}(m))$ to the mix M
- M verifies the signature of y_i and stores $\langle \ell_i, x_i, y_i \rangle$ and $\langle \ell_i, n_i \rangle$ for number ℓ_i

Phase 3*:

- After collecting all votes, M publishes every $\langle \ell_i, x_i, y_i \rangle$ in random order
- V_i verifies $\langle \ell_i, x_i, y_i \rangle$ was published and sends $\mathsf{aenc}(\langle \ell_i, n_i, k_i \rangle, r'_i, \mathsf{pk}(m))$ to M
- M verifies the pair $\langle \ell_i, n_i \rangle$ and then opens x_i using k_i to retrieve v_i and validates it is a well-formed vote
- Once all votes are collected the votes are released in random order

To protect against flooding attacks, as described in the previous section, our mix-network implementation authenticates the messages it receives. Only valid messages from valid participants in the protocol are allowed to pass through the anonymous channel. In Phase 2^*, this validation is enforced by verifying that messages are signed by the administrator. Although not mentioned in the original paper, it is important that these signed messages are also checked for distinctness. In other words, a voter should not be able to send multiple copies of its administrator signed vote through the mix. The absence of this requirement is a bug in the original protocol design.

In Phase 3^*, mix inputs are validated through the nonce n_i, which can only be mutually held by M and V_i, provided voter V_i supplied a valid mix input in the preceding phase. Notice that the flushing operation from Phase 2^* and Phase 3^* is triggered only after all inputs have been collected and validated. In particular, it is crucial that the mix from Phase 3^* validates that x_i and k_i produce a well formed vote before releasing the results.

Beyond the protocol modifications that enable the mixes to authenticate messages, our version of the protocol also requires that communication with the anonymous channel is encrypted. Also note that the mixes in our protocol actually assume the role of the collector. If so desired, the two parties can easily be separated to decentralize trust. Using SPAN, we have validated, in the symbolic model, that our changes result in a protocol that preserves vote privacy.

3.5 PRÊT À VOTER

Prêt à Voter [14] is a mix-network based voting protocol that provides a simple and intuitive mechanism by which a set of voters $(V_1, \ldots, V_n)$ can carry out elections with the help of a set of tellers $(T_1, \ldots, T_k)$ and an election authority (A). Each teller has two public key pairs. Using these keys and a set of random values, the authority creates a set of ballot forms with the following properties. Each ballot has two columns; the left column lists the candidates in random order and the right column provides space for a vote to be recorded. The bottom of the right column also holds an "onion" which encodes the ordering (cyclic offset) for the candidates on the left hand side of the ballot.

The precise construction of a ballot is as follows. The authority first generates a random seed,

$$\text{seed} := g_0, g_1, \ldots, g_{2k-1}$$

where each g_i (for $i \in \{1, \ldots, 2k-1\}$), called a germ, is drawn from an appropriately sized

field. For a candidate list of size v, the seed is used generate the cyclic offset

$$\theta := \sum_{i=0}^{2k-1} (d_i)(\mathsf{mod}\ v)$$

where $d_i := \mathsf{hash}(g_i)(\mathsf{mod}\ v)$. Each teller i has public keys $\mathsf{pk}(T_{2i})$ and $\mathsf{pk}(T_{2i-1})$ which are used to construct the onion

$$\{\langle g_{2k-1}, \{\langle g_{2k-1}, \ldots \{\langle g_0, D_0\rangle\}_{\mathsf{pk}(T_0)} \ldots\rangle\}_{\mathsf{pk}(T_{2k-2})}\rangle\}_{\mathsf{pk}(T_{2k-1})}$$

where D_0 is a nonce uniquely chosen for each onion. Each layer $D_{i+1} := \{\langle g_i, D_i\rangle\}_{\mathsf{pk}(T_i)}$ asymmetrically encrypts a germ and the previous layer of the onion.

The election authority generates a set of ballots whose count greatly exceeds the number of voters. To cast a vote, a voter authenticates with the authority after which a random ballot is chosen by the voter. In the voting booth, the voter marks his/her choice on the right hand side of the ballot and removes the left hand side for shredding. The values on the right side of the ballot (the vote position and onion) are read by a voting device and then retained by the voter as a receipt. Once read by the voting device, the values are passed to the tellers that manipulate pairs of the form $\langle r_{2i}, D_{2i}\rangle$. The first teller receives the pair $\langle r, D_{2k}\rangle$ where r is the vote position and D_{2k} is the onion. Upon receiving such a pair, each teller T_{i-1} performs the following operations.

- Apply the secret key $\mathsf{sk}(T_{2i-1})$ to D_{2i} to reveal the germ g_{2i-1} and the next layer of the onion D_{2i-1}.
- Recover $d_{2i-1} = \mathsf{hash}(g_{2i-1})(\mathsf{mod}\ v)$ and obtain $r_{2i-1} = (r_{2i} - d_{2i-1})(\mathsf{mod}\ v)$.
- Form the new pair $\langle r_{2i-1}, D_{2i-1}\rangle$.

After applying this transformation for each pair in the batch it receives, teller T_{i-1} performs a secret shuffle on the resulting transformed pairs. Teller T_{i-1} then repeats this process on the shuffled values using its second secret key $\mathsf{sk}(T_{2i-2})$ to obtain a new set of pairs with the form $\langle r_{2i-2}, D_{2i-2}\rangle$. These pairs are shuffled again and then passed to the next teller T_{i-2}. The output of the last teller is the value of r_0 which identifies a voter's vote.

Our analysis of this version of the Prêt à Voter protocol has uncovered a previously unknown flaw in the protocol's design. The error arises from the assumption that the elements of the field from which the germs are drawn are evenly distributed when their hash is taken modulo v. To understand this error in more detail, let us consider the simple case

when there are two candidates (0 and 1) and one teller. Let F be a field with M elements and

$$F_j = \{g \mid g \in F \text{ and } \mathsf{hash}(g)(\mathsf{mod}\ 2) = j\}$$

for $j \in \{0, 1\}$. There is no guarantee that $F_0 = F_1$ and thus the probability of the two cyclic offsets $\theta_0 = (\frac{F_0}{F})(\frac{F_0}{F}) + (\frac{F_1}{F})(\frac{F_1}{F})$ and $\theta_1 = 2(\frac{F_0}{F})(\frac{F_1}{F})$ in the randomly chosen ballots may be different. This can give an attacker an advantage in attempting to infer a vote from a ballot receipt. To fix this issue, the hash function should be replaced by a pseudo-random function [83]. In Section 4.7 we demonstrate how SPAN can be used to help identify and fix this bug.

CHAPTER 4: THE STOCHASTIC PROTOCOL ANALYZER

In this chapter, we introduce the first techniques for automated (symbolic) analysis of randomized security protocols. In particular, we propose one algorithm for analyzing state-based safety properties and two algorithms for analyzing indistinguishability in randomized protocols. Our techniques are implemented in the Stochastic Protocol ANalyzer (SPAN). As highlighted in Chapter 2, algorithms for analyzing randomized protocols require techniques for solving state-based safety and indistinguishability properties in POMDPs.

Safety properties We begin this chapter by presenting a technique (Algorithm 4.1 on page 32) for analyzing state-based safety properties of POMDPs. The algorithm is a based on a well-known technique in the verification of POMDPs in which one translates a POMDP $\mathcal{M}$ into a fully observable MDP known as a belief MDP $\mathcal{B}(\mathcal{M})$. Properties of $\mathcal{M}$ can then be derived from an analysis of $\mathcal{B}(\mathcal{M})$.

Indistinguishability properties In Section 4.2 we study the problem of deciding indistinguishability in POMDPs. Our first result (Theorem 4.2 on page 32) shows indistinguishability of finite POMDPs is in **NC** by a reduction of POMDP indistinguishability to equivalence in PFAs, which is known to be in **NC** (and hence also in **P**) [84, 85, 86, 87]. Further, we show that POMDP indistinguishability is at least as hard as PFA equivalence. The hardness result continues to hold for acyclic POMDPs, where an acyclic POMDP is a POMDP that has a set of "final" absorbing states and the only cycles in the underlying graph are self-loops on these states.

For acyclic finite POMDPs, we present another algorithm for checking indistinguishability based on a technique that translates a POMDP $\mathcal{M}$ into a fully observable MDP $\mathcal{B}(\mathcal{M})$, known as the belief MDP of $\mathcal{M}$. It was shown in [88] that two POMDPs are indistinguishable if and only if the belief MDPs they induce are bisimilar as labeled MDPs. When $\mathcal{M}$ is finite and acyclic and then its belief MDP $\mathcal{B}(\mathcal{M})$ is finite and acyclic and its bisimulation relation can be checked inductively (Algorithm 4.2 on page 34).

Complexity results Protocols in SPAN are described by a finite set of roles (agents) that interact asynchronously by passing messages. Each role models an agent in a protocol session and hence we only consider bounded number of sessions. An action in a role performs either a message input, or a message output or a test on messages. The adversary schedules the order in which these actions are executed and generates input recipes comprised of public information and messages previously output by the agents. In general, there are

an unbounded number of input recipes available at each input step, resulting in POMDPs that are infinitely branching. SPAN, however, searches for bounded attacks by bounding the size of attacker messages. Under this assumption, protocols give rise to finite acyclic POMDPs. Even with this assumption, protocols specified in SPAN describe POMDPs that are exponentially larger than their description. Nevertheless, we show that when considering protocols defined over subterm convergent equational theories, indistinguishability of randomized security protocols is in **PSPACE** for bounded Dolev-Yao adversaries. We further show that the problem is at least as hard as $\#\mathsf{SAT}_{\mathsf{D}}$ and hence it is both **NP**-hard and **coNP**-hard.

Tool overview The main engine of SPAN translates a randomized security protocol into an acyclic finite POMDP by recursively unrolling all protocol executions and grouping states according to those that are indistinguishable. We implemented two algorithms for checking indistinguishability in SPAN. The first algorithm, called the PFA algorithm, checks indistinguishability of P and P' by converting them to corresponding PFAs A and A' as in the proof of decidability of indistinguishability of finite POMDPs. PFA equivalence can then be solved through a reduction to linear programming [85]. The second algorithm, called the on-the-fly (OTF) algorithm, is based on the technique of checking bisimulation of belief MDPs. Although asymptotically less efficient than the PFA algorithm, the recursive procedure for checking bisimulation in belief MDPs can be embedded into the main engine of SPAN with little overhead, allowing one to analyze indistinguishability on-the-fly as the POMDP models are constructed.

Evaluation and case studies In this chapter, we also evaluate the model checking algorithms implemented in SPAN. In this evaluation process, we conduct the first automated symbolic analysis for several new classes of security protocols including mix networks [9], dinning cryptographers networks [80, 41] a 3-ballot electronic voting protocol [17], the FOO electronic voting protocol [43] and Prêt à Voter [14]. In our analysis, we uncovered several new security vulnerabilities in electronic voting protocols. When analyzing the FOO voting protocol, we uncovered a replay attack. In addition, we discovered that realizing perfectly anonymous channels in the FOO protocol requires non-trivial modification to the protocol design, which if not done carefully, can lead to errors. A bug in the design of the Prêt à Voter protocol was also uncovered.

4.1 SAFETY PROPERTIES

In the analysis of state-based safety properties in POMDPs, the fundamental challenge is to find an attacker for a given POMDP that maximizes the probability of reaching a set of target (bad) states. Unfortunately, techniques for solving reachability problems in POMDPs are far less efficient than those for MDPs, the fully observable counterpart to POMDPs (where attackers are a function from *executions* to actions). The reason for the added complexity is that at any given point in the execution of a POMDP, the attacker can only view the observation generated by a state and can thus only infer a distribution over the possible states. Further, an attacker for a POMDP needs to define a consistent strategy across all executions that produce the same sequence of observations. The actions chosen in one branch of an execution may affect the actions that can be made in another branch of an execution. By contrast, when trying to maximize the probability of reaching a target set of states in an MDP, it is possible to make a local decision about which action maximizes the probability of reaching the target states.

Several results [89, 22] corroborate this story, showing that many key verification problems for POMDPs are undecidable. Although various solution techniques have been proposed [90], and there have been successful applications to AI and planning [91], tractable reasoning about POMDPs typically relies on approximation techniques or simplifications to the model (discounts). Complicating matters further, randomized security protocols induce POMDPs that are infinitely branching. At every transition corresponding to protocol input, an infinite number of possible recipes can be supplied by a Dolev−Yao attacker. Taming the state space explosion that results from this infinite branching on inputs is a huge challenge, even in the non-randomized case. We adopt the philosophy of the SATMC [5] tool, in that we will search for bounded attacks. That is, our tool answers the question; for a given input recipe depth k, what is the maximum probability of reaching a set of target states? The assumption of bounded recipe depth allows randomized security protocol to be modeled by POMDPs that are finite branching.

One of the most successful techniques in the approximation of optimal attackers for POMDPs is to translate a POMDP $\mathcal{M}$ into a fully observable belief MDP $\mathcal{B}(\mathcal{M})$ that emulates it. One can then analyze $\mathcal{B}(\mathcal{M})$ to infer properties of $\mathcal{M}$. The states of $\mathcal{B}(\mathcal{M})$ are probability distributions over the states of $\mathcal{M}$. Further, given a state b of $\mathcal{B}(\mathcal{M})$, if states z_1, z_2 of $\mathcal{M}$ are such that $b(z_1), b(z_2)$ are non-zero then z_1 and z_2 must have the same observation. Hence, by abuse of notation, we can define $\mathsf{obs}(b)$ to be $\mathsf{obs}(z)$ if $b(z) \neq 0$. Intuitively, an execution $\rho = b_0 \xrightarrow{\alpha_1} b_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} b_m$ of $\mathcal{B}(\mathcal{M})$ corresponds to the set of all executions ρ' of $\mathcal{M}$ such that $\mathsf{tr}(\rho') = \mathsf{obs}(b_0)\alpha_1\mathsf{obs}(b_1)\alpha_2 \cdots \alpha_m\mathsf{obs}(b_m)$. The measure of execution ρ in

$\mathcal{B}(\mathcal{M})$ is exactly $\mathsf{prob}_{\mathcal{M}}(\mathsf{obs}(b_0)\alpha_1\mathsf{obs}(b_1)\alpha_2 \cdots \alpha_m\mathsf{obs}(b_m))$.

The initial state of $\mathcal{B}(\mathcal{M})$ is the distribution that assigns 1 to the initial state of $\mathcal{M}$. Intuitively, on a given state b of $\mathcal{B}(\mathcal{M})$ and an action α, there is at most one successor state $b^{\alpha,o}$ for each observation o. The probability of transitioning from b to $b^{\alpha,o}$ is the probability that o is observed given that the distribution on the states of $\mathcal{M}$ is b and action α is performed; $b^{\alpha,o}(z)$ is the conditional probability that the actual state of the POMDP is z. The formal definition follows.

Definition 4.1 *Let $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ be a POMDP. The belief MDP of $\mathcal{M}$, denoted $\mathcal{B}(\mathcal{M})$, is the tuple $(\mathsf{Dist}(Z), \delta_{z_s}, \mathsf{Act}, \Delta^{\mathcal{B}})$ where $\Delta^{\mathcal{B}}$ is defined as follows. For $b \in \mathsf{Dist}(Z)$, action $\alpha \in \mathsf{Act}$ and $o \in \mathcal{O}$, let*

$$p_{b,\alpha,o} = \sum_{z \in Z} b(z) \cdot \left(\sum_{z' \in Z \wedge \mathsf{obs}(z') = o} \Delta(z, \alpha)(z') \right).$$

$\Delta^{\mathcal{B}}(b, \alpha)$ is the unique distribution such that for each $o \in \mathcal{O}$, if $p_{b,\alpha,o} \neq 0$ then $\Delta^{\mathcal{B}}(b, \alpha)(b^{\alpha,o}) = p_{b,\alpha,o}$ where for all $z' \in Z$,

$$b^{\alpha,o}(z') = \begin{cases} \frac{\sum_{z \in Z} b(z) \cdot \Delta(z,\alpha)(z')}{p_{b,\alpha,o}} & \text{if } \mathsf{obs}(z') = o \\ 0 & \text{otherwise} \end{cases}.$$

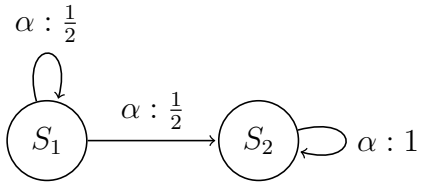
This definition results in a correspondence between the maximal reachability probabilities in a POMDP $\mathcal{M}$ and the belief MDP $\mathcal{B}(\mathcal{M})$ it induces. The following proposition, due to Norman et al. [92], makes this correspondence precise. In the result below, for a POMDP (resp. MDP) $\mathcal{M}$ and a set of observations O (resp. states T), we write $\mathsf{prob}^{max}_{\mathcal{M}}(O)$ (resp. $\mathsf{prob}^{max}_{\mathcal{M}}(T)$) to denote the maximum probability with which $\mathcal{M}^A$ reaches states with observations in O (resp. states from T) for any adversary $\mathcal{A}$.

Proposition 4.1 *Let $\mathcal{M} = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ be a POMDP, $O \subseteq \mathcal{O}$ and $T_O = \{b \in \mathsf{Dist}(Z) \mid \forall z \in Z.(b(z) > 0 \Rightarrow \mathsf{obs}(z) \in O)\}$. Then $\mathsf{prob}^{max}_{\mathcal{M}}(O) = \mathsf{prob}^{max}_{\mathcal{B}(\mathcal{M})}(T_O)$.*

In general, belief MDPs are defined over a continuous state space; even simple POMDP models can yield an infinite number of distributions on states. For instance, see the POMDP from Example 4.1. It is this continuous state space that makes belief MDPs difficult to analyze. Fortunately, the calculus from Section 2.7 doesn't include an operator for replication, as we are mainly interested in single session protocols like those from Chapter 3. This means that protocol executions are of a fixed length and can be encoded as acyclic POMDPs that reach a set of finite absorbing states after a bounded number of actions. However, even

for acyclic POMDPs, the number of reachable belief states can grow much larger than the number of states in the original POMDP. Again, see example 4.1.

Example 4.1 *Consider the POMDP* $\mathcal{M}$



where $Z = \{S_1, S_2\}$, $z_s = A$, $\mathsf{Act} = \{\alpha\}$ *and* $\mathsf{obs}(S_1) = \mathsf{obs}(S_2)$. *Observe that any belief state of the form* $\{S_1 : \frac{1}{2^k}, S_2 : 1 - \frac{1}{2^k}\}$ *is reachable in* k *steps in* $\mathcal{B}(\mathcal{M})$. *Clearly, the number of reachable states in* $\mathcal{B}(\mathcal{M})$ *grows with the length of executions is not finite.*

Let Q be a randomized security protocol such that $[\![Q]\!] = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$. Define $[\![Q_d]\!] = (Z, z_s, \mathsf{Act}_d, \Delta_d, \mathcal{O}, \mathsf{obs})$ where every $\alpha \in \mathsf{Act}_d$ is such that $\mathsf{depth}(\alpha) \leq d$ and for all $z \in Z$, $\Delta_d(z, \alpha) = \Delta(z, \alpha)$ if $\alpha \in \mathsf{Act}_d$ and otherwise $\Delta_d(z, \alpha)$ is undefined. For a security protocol Q, probability p and safety property ψ, the *bounded model checking problem* for depth d is to determine if $[\![Q_d]\!] \models_p \psi$. As described above, $[\![Q_d]\!]$ can be translated into a finite acyclic fully observable belief MDP $\mathcal{B}([\![Q_d]\!])$. By analyzing $\mathcal{B}([\![Q_d]\!])$, one can generate an attacker for $[\![Q_d]\!]$ that optimizes the probability of reaching a target set of states $Z \setminus \psi$. These optimal reachability probabilities can be computed using Algorithm 4.1, where we assume a finite set of absorbing states B_{abs}. The algorithm works by recursively computing the maximum probability of attack by exploring states in a depth first way. Such an approach can avoid exploring many redundant portions of the state space.

The correctness of our algorithm, which follows from Proposition 4.1, is given below.

Theorem 4.1 *Let* Q *be a protocol and* $d \in \mathbb{N}$ *be such that* $[\![Q_d]\!] = (Z, z_s, \mathsf{Act}_d, \Delta, \mathcal{O}, \mathsf{obs})$. *For a given probability* p *and state-based safety property* $\psi \subseteq Z$, *if* $[\![Q_d]\!] \models_p \psi$ *iff* $\textsc{maxAttack}(\delta_z, Z \setminus \psi) \leq 1 - p$ *for the belief MDP* $\mathcal{B}([\![Q_d]\!])$.

4.2 INDISTINGUISHABILITY PROPERTIES

In this section, we study the problem of deciding indistinguishability properties in POMDPs. The techniques we develop for analyzing POMDPs will provide the foundation for the indistinguishability algorithms we implement in the Span protocol analysis tool. In our first result, we show that POMDP indistinguishability is in **NC** (and hence also **P**) by a reduction to PFA equivalence, also known to be in **NC** [84, 85, 86, 87]. All of the results in this section assume finite POMDPs.

Algorithm 4.1 *On-the-fly model checking of safety properties in finite-length belief MDPs.*

```
1: procedure MAXATTACK(beliefState b, targetStates T)
2:     p ← 0
3:     if b ∈ B_abs then
4:         for z ∈ support(b) do
5:             if z ∈ T then
6:                 p ← p+b(z)
7:         return p
8:     for α ∈ Act do
9:         for o ∈ O do
10:            p ← max(p, MAXATTACK(b^{α,o}, T))
11:            if p == 1 then
12:                return 1
13:    return p
```

Theorem 4.2 *Indistinguishability of finite POMDPs is in* **NC**.

Proof. Consider two POMDPs $\mathcal{M}_i = (Z_i, z_s^i, \mathsf{Act}_i, \Delta_i, \mathcal{O}_i, \mathsf{obs}_i)$ for $i \in \{0,1\}$ with the same set of actions Act and observations $\mathcal{O}$. We shall construct PFAs A_0 and A_1 such that $\mathcal{M}_0 \approx \mathcal{M}_1$ iff $\mathsf{A}_0 \equiv \mathsf{A}_1$ as follows. For $i \in \{0,1\}$, let "bad_i" be a new state and define the PFA $\mathsf{A}_i = (Q_i, \Sigma, q_s^i, \Delta_i', F_i)$ where $Q_i = Z_i \cup \{\mathsf{bad}_i\}$, $\Sigma = (\mathsf{Act} \times \mathcal{O})$, $q_s^i = z_s^i$, $F_i = Z_i$ and Δ_i' defined is as follows.

$$\Delta_i'(q,(\alpha,o))(q') = \begin{cases} \Delta_i(q,\alpha)(q') & \text{if } q,q' \in Z_i \text{ and } \mathsf{obs}(q) = o \\ 1 & \text{if } q \in Z_i,\ \mathsf{obs}(q) \neq o \text{ and } q' = \mathsf{bad}_i \\ 1 & \text{if } q,q' = \mathsf{bad}_i \\ 0 & \text{otherwise} \end{cases}.$$

Let $u = (\alpha_1, o_0)\dots(\alpha_k, o_{k-1})$ be a non-empty word on Σ. To the word u, associate the trace $\overline{o}_u = o_0\alpha_1 o_1 \alpha_2 \dots \alpha_{k-1} o_{k-1}$ of $\mathcal{M}$. Observe that a run $\rho \in z_0 \dots z_{k-1} z_k$ of A_i on u is an accepting run if and only the following hold.

1. $z_j \in Z$ for each $0 \leq j \leq k$
2. $\mathsf{obs}(z_j) = o_j$ for each $0 \leq j < k$
3. $\Delta'(z_j, (\alpha_{j+1}, o_i))(z_{j+1}) = \Delta(z_j, \alpha_{j+1})(z_{j+1}) > 0$ for each $0 \leq j < k$

From these observations, it is easy to see that $\mathsf{prob}_{\mathsf{A}_i}(u)$

$$= \sum_{\substack{z_j \in Z,\ \forall\, 0 \le j < k,\\ \mathsf{obs}(z_j) = o_j,\ z \in Z}} \Delta_i(z_0, \alpha_1)(z_1) \ldots \Delta_i(z_{k-2}, \alpha_{k-1})(z_{k-1}) \Delta_i(z_{k-1}, \alpha_k)(z)$$

$$= \sum_{\substack{z_j \in Z,\ \forall\, 0 \le j < k,\\ \mathsf{obs}(z_j) = o_j}} \sum_{z \in Z} \Delta_i(z_0, \alpha_1)(z_1) \ldots \Delta_i(z_{k-2}, \alpha_{k-1})(z_{k-1}) \Delta_i(z_{k-1}, \alpha_k)(z)$$

$$= \sum_{\substack{z_j \in Z,\ \forall\, 0 \le j < k,\\ \mathsf{obs}(z_j) = o_j}} \Delta_i(z_0, \alpha_1)(z_1) \ldots \Delta_i(z_{k-2}, \alpha_{k-1})(z_{k-1}) \sum_{z \in Z} \Delta_i(z_{k-1}, \alpha_k)(z)$$

$$= \sum_{\substack{z_j \in Z,\ \forall\, 0 \le j < k,\\ \mathsf{obs}(z_j) = o_j}} \Delta_i(z_0, \alpha_1)(z_1) \ldots \Delta_i(z_{k-2}, \alpha_{k-1})(z_{k-1}) \cdot 1$$

$$= \mathsf{prob}_{\mathcal{M}_i}(\overline{o}_u).$$

The proposition follows immediately from the fact that $\mathsf{prob}_{\mathsf{A}_i}(u) = \mathsf{prob}_{\mathcal{M}_i}(\overline{o}_u)$.

Theorem 4.3 *Indistinguishability of finite acyclic POMDPs is at least as hard as PFA equivalence.*

Proof. Let $\mathsf{A}_i = (Q_i, \Sigma, q_s^i, \Delta_i, F_i)$ for $i \in \{0, 1\}$ be PFAs such that $|Q_0| = m$ and $|Q_1| = n$. It was shown in [93] that A_0 and A_1 are distinguishable iff they are distinguishable on words of length at most $\ell = m + n$. For $i \in \{0, 1\}$, let $\mathcal{M}_i$ be the POMDP $(Z_i, z_s^i, \mathsf{Act}_i, \Delta_i', \mathcal{O}_i, \mathsf{obs}_i)$ where $Z_i = \{(q, j) \mid q \in Q_i \text{ and } 0 \le j \le \ell\} \cup \{\mathsf{accept}_i, \mathsf{dead}_i\}$, $z_s^i = (q_s^i, 0)$, $\mathsf{Act}_i = \Sigma_i \cup \{\tau\}$, $\mathcal{O}_i = \{o_1, o_2\}$, $\mathsf{obs}_i(\mathsf{accept}_i) = o_2$, $\mathsf{obs}_i(z) = o_1$ for all $z \in Z_i \setminus \{\mathsf{accept}_i\}$ and Δ_i' is as follows. For all $\alpha \in \mathsf{Act}$,

$$\Delta_i'(z, \alpha)(z') = \begin{cases} \Delta_i(q, \alpha)(q') & \text{if } z = (q, j-1),\ z' = (q', j), \alpha \in \Sigma, j \in \{0, \ldots, \ell\} \\ 1 & \text{if } z = (q, \ell),\ \alpha \in \Sigma, z' = \mathsf{dead}_i \\ 1 & \text{if } z = (q, j),\ \alpha = \tau,\ q \in F_i, z' = \mathsf{accept}_i \\ 1 & \text{if } z = (q, j),\ \alpha = \tau,\ q \notin F_i, z' = \mathsf{dead}_i \\ 1 & \text{if } z = z' = \mathsf{accept}_i \text{ or } z = z' = \mathsf{dead}_i \\ 0 & \text{otherwise.} \end{cases}$$

It is easy to see that $\mathcal{M}_0, \mathcal{M}_1$ are acyclic and that $\mathsf{A}_0, \mathsf{A}_1$ are equivalent iff $\mathcal{M}_0, \mathcal{M}_1$ are indistinguishable.

In Theorem 4.2, we presented a new algorithm for POMDP indistinguishability. As dicussed in Section 4.1, POMDPs can also be analyzed by a translation to beleif MDPs. Let $\mathcal{M}_i = (Z_i, z_s^i, \mathsf{Act}, \Delta_i, \mathcal{O}, \mathsf{obs}_i)$ for $i \in \{0, 1\}$ be POMDPs with the same set of actions and observations. In [88] the authors show that $\mathcal{M}_0$ and $\mathcal{M}_1$ are indistinguishable if and only if the beliefs $\delta_{z_s^0}$ and $\delta_{z_s^1}$ are *strongly belief bisimilar*. Strong belief bisimilarity coincides with the notion of bisimilarity of labeled MDPs: a pair of states $(b_0, b_1) \in \mathsf{Dist}(Z_0) \times \mathsf{Dist}(Z_1)$ is said to have a strong belief bisimulation if (i) $\mathsf{obs}(b_0) = \mathsf{obs}(b_1)$, (ii) for all $\alpha \in \mathsf{Act}, o \in \mathcal{O}$, $p_{b_0,\alpha,o} = p_{b_1,\alpha,o}$ and (iii) the pair $(b_0^{\alpha,o}, b_1^{\alpha,o})$ has a strong belief bisimulation if $p_{b_0,\alpha,o} = p_{b_1,\alpha,o} > 0$. The states $(b_0, b_1) \in \mathsf{Dist}(Z_0) \times \mathsf{Dist}(Z_1)$ are said to be strongly belief bisimilar if there exists some strong belief bisimulation over (b_0, b_1). Observe that, in general, belief MDPs are defined over an infinite state space. It is easy to see that, for a finite acyclic POMDP $\mathcal{M}$, $\mathcal{B}(\mathcal{M})$ is acyclic and has a finite number of reachable belief states. Let $\mathcal{M}_0$ and $\mathcal{M}_1$ be as above and assume further that $\mathcal{M}_0, \mathcal{M}_1$ are finite and acyclic with absorbing states $Z_{\mathsf{abs}} \subseteq Z_0 \cup Z_1$. As a consequence of the result from [88] and the observations above, we can determine if two states $(b_0, b_1) \in \mathsf{Dist}(Z_0) \times \mathsf{Dist}(Z_1)$ are strongly belief bisimilar using the on-the-fly procedure from Algorithm 4.2.

Algorithm 4.2 On-the-fly bisimulation for finite acyclic POMDPs

```
1: function BISIMILAR(beliefState b_0, beliefState b_1)
2:     if obs(b_0) ≠ obs(b_1) then return false
3:     if support(b_0) ∪ support(b_1) ⊆ Z_abs then return true
4:     for α ∈ Act do
5:         for o ∈ O do
6:             if p_{b_0,α,o} ≠ p_{b_1,α,o} then return false
7:             if p_{b_0,α,o} > 0 and !BISIMILAR(b_0^{α,o}, b_1^{α,o}) then return false
8:     return true
```

4.3 COMPLEXITY OF INDISTINGUISHABILTY

In this section, we study the problem of deciding when two protocols are indistinguishable. Let P be a protocol such that $[\![P]\!] = (Z, z_s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$. Define $[\![P]\!]_d$ to be the POMDP $[\![P]\!]_d = (Z, z_s, \mathsf{Act}_d, \Delta, \mathcal{O}, \mathsf{obs})$ where $\mathsf{Act}_d \subseteq \mathsf{Act}$ is such that every $\alpha \in \mathsf{Act}$ is such that $\mathsf{depth}(\alpha) \leq d$. For a constant $d \in \mathbb{N}$, we define $\mathsf{InDist}(d)$ to be the decision problem that, given a subterm convergent equational theory $(\mathcal{F}, E)$ and protocols P, P' over $(\mathcal{F}, E)$, determines if $[\![P]\!]_d$ and $[\![P']\!]_d$ are indistinguishable. An equational theory $(\mathcal{F}, E)$ is *subterm convergent* [68] if for every equation $u = v \in E$ oriented as a rewrite rule $u \to v$, either v is a proper

subterm of u or v is a constant.

Theorem 4.4 *For a subterm convergent equational theory $(\mathcal{F}, E)$ and a constant $d \in \mathbb{N}$, $\mathsf{InDist}(d)$ is in* **PSPACE**.

Proof. Let P, P' be protocols of length ℓ. We give an algorithm for deciding $\mathsf{InDist}(d)$ as follows. Guess a trace $\overline{o} = o_0\alpha_1 o_1 \dots \alpha_n o_n$ such that $\alpha_i \in \mathsf{Act}_d$, where each observation is represented by a frame, and calculate the measure γ (resp. γ') of $\overline{o}$ in P_d (resp. P'_d) by enumerating the exponential number of executions. For each $\rho \in \mathsf{Exec}([\![P_d]\!])$ (resp. $\rho \in \mathsf{Exec}([\![P'_d]\!])$), check if ρ has trace $\overline{o}$ and if so, add the measure of ρ to γ (resp. γ'). Finally, check that $\gamma \neq \gamma'$. Because static equivalence is decidable in polynomial time for sub-term convergent theories [68], we can check if a given execution has trace $\overline{o}$ in polynomial time. We need only show that $\overline{o}$ can be guessed in polynomial space, every ρ can be enumerated in polynomial space and every execution has a polynomial measure.

Let s be the maximum arity of the function symbols in $\mathcal{F}$. Observe that every $(r, j) \in \mathsf{Act}_d$ for $j \in \{1, \dots, n\}$ is such that $|r| \leq s^{d+1}$. Let θ be dag-size of the largest term in the specification of protocols P and P'. Further let $\rho \in \mathsf{Exec}([\![P_d]\!])$ (resp. $\rho \in \mathsf{Exec}([\![P'_d]\!])$) where $\mathsf{last}(\rho) = (Q, \varphi, \sigma)$. We claim that, for any $u \in \mathsf{ran}(\varphi) \cup \mathsf{ran}(\sigma)$, $|u| \leq \ell(\theta + s^{d+1})$. Indeed, consider any execution $\rho \in \mathsf{Exec}([\![P_d]\!])$ (resp. $\rho \in \mathsf{Exec}([\![P'_d]\!])$) with m inputs and n outputs where $\mathsf{last}(\rho) = (Q, \varphi, \sigma)$, $\mathsf{dom}(\varphi) = \{w_1, \dots, w_n\}$ and $\mathsf{dom}(\sigma) = \{x_1, \dots, x_m\}$. The following are true.

1. $w_j\varphi = u_j(x_1, \dots, x_m)$ is an output term over $x_1, \dots x_m$ for all $j \in \{1, \dots, m\}$
2. $x_i\sigma = r_i(w_1, \dots, w_n)$ is a recipe over $w_1, \dots, w_n$ for all $i \in \{1, \dots, n\}$
3. $|w_0| + \dots + |w_n| = |u_0| + \dots + |u_n| + |x_1|, \dots, |x_m|$
4. $|x_1| + \dots + |x_m| = |r_0| + \dots + |r_m| + |w_0| \dots + |w_n|$

It follows that

$$\begin{aligned} |w_j|, |x_i| &\leq |u_0| + \dots + |u_n| + |r_0| + \dots + |r_m| \\ &\leq n \cdot \theta + m \cdot s^{d+1} \\ &\leq \ell \cdot \theta + \ell \cdot s^{d+1} \\ &\leq \ell(\theta + s^{d+1}). \end{aligned}$$

From the derivation above, we have that for any execution of length a most ℓ, $|\varphi|, |\sigma|$ are polynomially bounded. To see that the measure of an execution is polynomial, let $p = \mathsf{max}\{y \mid \frac{x}{y}$ is a transition probability in $P, P'\}$. Because the measure of an execution $\frac{x}{y}$ lies in [0,1] and $y \leq p^\ell$, we have $0 \leq x, y \leq p^\ell$.

We now give a lower bound for $\mathsf{InDist}(d)$ by a reduction from $\#\mathsf{SAT}_{\mathsf{D}}$. Recall that $\#\mathsf{SAT}_{\mathsf{D}}$ is the decision problem that, given a 3CNF formula ϕ and a constant $k \in \mathbb{N}$, checks if the number of satisfying assignments of ϕ is equal to k. In the proof of Theorem 4.5 (below), we will make use of a conditional operation

$$\mathsf{IF}\ [c_1 \wedge \ldots \wedge c_k]\ \mathsf{THEN}\ \mathtt{out}(u_0 +_p u_1)\ \mathsf{ELSE}\ \mathtt{out}(u'_0 +_{p'} u'_1)$$

that performs $\mathtt{out}(u_0 +_p u_1)$ if $[c_1 \wedge \ldots \wedge c_k]$ holds and otherwise performs $\mathtt{out}(u'_0 +_{p'} u'_1)$. This operation was omitted from the process calculus in Section 2.7 to simplify the presentation of results in Chapters 5 and 6. However, it should be considered a part of the process calculus of this chapter and is implemented in SPAN (see Section 4.4).

Theorem 4.5 *There exists a $d_0 \in \mathbb{N}$ such that $\#\mathsf{SAT}_{\mathsf{D}}$ reduces to $\mathsf{InDist}(d)$ in logspace for every $d > d_0$.*

Proof. Let ϕ be a 3CNF formula over the variables $x_0, \ldots, x_n$ with clauses $c_0, \ldots, c_m$. For a clause $c = (a_0 \vee a_1 \vee a_2)$ and a position $p \in \{0, 1, 2\}$, let $c^p = a_p$. We define protocols P, P' such that ϕ has k satisfying assignments iff P and P' are indistinguishable. The equational theory of the protocol will have private names $k_0, \ldots, k_n, s_0, \ldots, s_m \in \mathcal{N}_{\mathsf{priv}}$, public names $\{\top, \bot\} \in \mathcal{N}_{\mathsf{pub}}$, signature $\mathcal{F} = \{\mathsf{sdec}, \mathsf{senc}, \mathsf{conj}, \mathsf{neg}\}$ and equations $E = \{\mathsf{sdec}(\mathsf{senc}(x_0, x_1), x_1) = x_0$, $\mathsf{neg}(\top) = \bot$, $\mathsf{neg}(\bot) = \top$, $\mathsf{conj}(\bot, \bot, \bot) = \bot$, $\mathsf{conj}(\top, x_0, x_1) = \top$, $\mathsf{conj}(x_0, \top, x_1) = \top$, $\mathsf{conj}(x_0, x_1, \top) = \top\}$. Let $key(c_j^p) = k_i$ if c_j^p is the (negation of) a variable x_i. For each $i \in \{1, \ldots, n\}$, define $Q_i = \mathtt{out}(\mathsf{senc}(\top, k_i) +_{\frac{1}{2}} \mathsf{senc}(\bot, k_i))$ and for each $j \in \{1, \ldots, m\}$ define

$$\begin{aligned} C_j \quad = \quad & \mathtt{in}(y_j^0) \cdot \mathtt{in}(y_j^1) \cdot \mathtt{in}(y_j^2) \cdot [\mathsf{senc}(\mathsf{sdec}(y_j^0, key(c_j^0)), key(c_j^0)) = y_j^0 \wedge \ldots \wedge \\ & \mathsf{senc}(\mathsf{sdec}(y_j^2, key(c_j^2)), key(c_j^2)) = y_j^2] \cdot \mathtt{out}(\mathsf{senc}(\mathsf{conj}(w_j^0, w_j^1, w_j^2), s_j)) \end{aligned}$$

where $w_j^p = \mathsf{neg}(\mathsf{sdec}(y_j^0, key(c_j^0)))$ if c_j^p is the negation of a variable and $w_j^p = \mathsf{sdec}(y_j^0, key(c_j^0))$ otherwise. Let

$$V_0 := \mathtt{in}(z_0) \cdot \ldots \cdot \mathtt{in}(z_m) \cdot [\mathsf{senc}(\mathsf{sdec}(z_0, s_0), s_0) = z_0 \wedge \ldots \wedge \mathsf{senc}(\mathsf{sdec}(z_m, s_m), s_m) = z_m].$$

Further let

$$V := \mathsf{IF}\ [\mathsf{sdec}(z_0, s_0) = \top \wedge \ldots \wedge \mathsf{sdec}(z_m, s_m) = \top]\ \mathsf{THEN}\ \mathtt{out}(\top)\ \mathsf{ELSE}\ \mathtt{out}(\bot)$$

and $V' = \mathtt{out}(\top +_{\frac{k}{2^n}} \bot)$. Finally, let $P_0 = Q_0 | \ldots | Q_n | C_0 | \ldots | C_m$, $P = P_0 | V_0 \cdot V$ and $P' = P_0 | V_0 \cdot V'$. Clearly P and P' only differ on executions that output $\top$ or $\bot$. Observe

that there exist traces $\overline{o}_\top$ and $\overline{o}_\bot$ such that, for every execution $\rho \in \mathsf{Exec}(P) \cup \mathsf{Exec}(P')$, if ρ outputs $\top$ then $\mathsf{tr}(\rho) = \overline{o}_\top$ and if ρ outputs $\bot$ then $\mathsf{tr}(\rho) = \overline{o}_\bot$. By construction, $\mathsf{prob}_{P'}(\overline{o}_\top) = \frac{k}{2^n}$ and $\mathsf{prob}_{P'}(\overline{o}_\bot) = 1 - \frac{k}{2^n}$. Further, if ϕ has exactly k truth assignments then $\mathsf{prob}_P(\overline{o}_\top) = \frac{k}{2^n}$ and $\mathsf{prob}_P(\overline{o}_\bot) = 1 - \frac{k}{2^n}$. The result follows.

From Theorem 4.5, we have the following.

Corollary 4.1 $\mathsf{InDist}(d)$ *is* ***NP****-hard and* ***coNP****-hard.*

4.4 PROTOCOL SPECIFICATION

The SPAN tool makes several enhancements to the core process calculus for randomized security protocols presented in Section 2.7. Protocols are specified over an equational theory that includes a user-defined sort system [2]. Sorts are used to distinguish between different types of data. One can also define sorts to be sub-sorts of other sorts, allowing a more fine-grained restriction of a data type.

Example 4.2 *Consider the protocol and equational theory* $(\mathcal{F}_{\mathsf{aenc}}, E_{\mathsf{aenc}})$ *from Example 2.3. One may specify a sort system with four sorts:* `msg`*,* `nonce`*,* `pkey` *and* `skey` *where the latter three sorts are sub-sorts of* `msg`*. The functions can then be assigned sorts as follows.*

$$\begin{aligned}
\mathsf{sk} &:= \mathit{nonce} \to \mathit{skey} \\
\mathsf{pk} &:= \mathit{nonce} \to \mathit{pkey} \\
\mathsf{aenc} &:= \mathit{msg} \times \mathit{nonce} \times \mathit{pkey} \to \mathit{msg} \\
\mathsf{adec} &:= \mathit{msg} \times \mathit{skey} \to \mathit{msg} \\
\mathsf{pair} &:= \mathit{msg} \times \mathit{msg} \to \mathit{msg} \\
\mathsf{fst}, \mathsf{snd} &:= \mathit{msg} \to \mathit{msg}
\end{aligned}$$

Roles are specified as a sequential composition of three basic constructs (actions). The interpretation of each action is a straightforward from the semantics presented in Figure 2.1, so we omit formal definitions in the descriptions below. Note that each action is required to be annotated with a *phase* number. For some protocols, their correctness relies on a strict ordering between different sets of events/operations. For example, the FOO protocol from Section 3.4 has 3 phases that must execute sequentially. Phase numbers are used as a mechanism to enforce these kind of orderings. All enabled actions with phase i must complete before any action from phase j for $j > i$ can begin.

Guarded outputs. A guarded output takes the form

$$[c_1 \wedge \ldots \wedge c_k]\ \mathtt{out}(p_1 \to \overline{u_1}\#R_1 + \ldots + p_\ell \to \overline{u_\ell}\#R_\ell)$$

where c_i (for $i \in \{1, \ldots, k\}$) are conjuncts of the form $\top$, $u = v$ or $u \neq v$ for $u, v \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathcal{X})$, $p_1, \ldots, p_\ell \in (0, 1]$ are such that $\sum_{i=1}^{k} p_i = 1$, $\overline{u_1}, \ldots, \overline{u_\ell}$ are sequences of terms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ and $R_1, \ldots, R_\ell$ are roles. If all of the conjuncts evaluate to `true` then the action outputs the sequence of terms $\overline{u_j}$ and places the actions from R_j to the head of the role with probability p_j for $j \in \{1, \ldots, \ell\}$. Otherwise the action blocks. To reduce the syntactic overhead for the output process, we allow it be replaced by $\mathtt{permute}(u_1, \ldots, u_\ell)$ which outputs each permutation of the set of terms $\{u_1, \ldots, u_\ell\}$ with probability $\frac{1}{2^k}$. Consider the mix-network from Example 2.3. The output operation could be replaced by the more succinct "$[\top]\ \mathtt{permute}(\mathsf{adec}(z_1, \mathsf{sk}(k_M)), \ldots, \mathsf{adec}(z_n, \mathsf{sk}(k_M))$".

Conditional outputs. A conditional output takes the form

$$\text{IF } [c_1 \wedge \ldots \wedge c_k] \text{ THEN } \mathtt{out}_1 \text{ ELSE } \mathtt{out}_2$$

where the conjuncts $c_1 \ldots c_k$ and output processes $\mathtt{out}_1$ and $\mathtt{out}_2$ take the same form as they do for guarded outputs. If all of the conjuncts evaluate to `true` then the actions executes $\mathtt{out}_1$ otherwise it executes $\mathtt{out}_2$.

Guarded inputs. A guarded input takes the form $\mathtt{in}(y\{u\})$ where y is a variable and $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$. The action takes an input v, generated from a recipe and the frame, and binds y to v. For v to be a valid input, it is required that there is a substitution θ with domain $\mathsf{vars}(u)$ such that $u\theta = v$. Note that θ must respect sorts, that is, if $y \in \mathsf{dom}(\theta)$ then y and $y\theta$ have the same sort. Essentially, this allows pattern matching on the symbols and sorts of input terms. Consider the protocol and sort system from Example 4.2. Let m and n be variables with sort `mes` and `nonce`, respectively. The pattern "$\mathsf{aenc}(m, n, \mathsf{pk}(k_M))$" requires recipes to generate input terms in which the root symbol is aenc where the first parameter has sort `mes`, the second parameter has sort `nonce` and the final parameter is the public key of the mix.

4.5 IMPLEMENTATION

In this section we describe the implementation of our model checking algorithms for indistinguishability and safety properties of randomized security protocols. The main engine of our tool, SPAN, translates a protocol into a POMDP, belief MDP or PFA by exploring all protocol executions and grouping states according to those that are statically equivalent. This engine, in conjunction with (the proof of) Theorem 4.2, allows SPAN to solve the indistinguishability problem for randomized security protocols as follows. For protocols P, P', translate $[\![P]\!], [\![P']\!]$ into PFAs A, A' and determine if $\mathsf{A} \equiv \mathsf{A}'$ using the linear programming algorithm from [85]. We will henceforth refer to this approach as the PFA algorithm and the approach from Algorithm 4.2 as the OTF algorithm. SPAN also implements the OTF approach from Algorithm 4.1 to analyze safety properties. The tool is implemented in Java and is available for download at [94]. In what follows, we describe the libraries and external engines that SPAN makes use of.

Static equivalence and deduction engines. Currently, SPAN supports two external engines for solving the static equivalence and deduction questions: KISS [95] and AKISS [96]. The KISS tool requires an equational theory $(\mathcal{F}, E)$ that is *sub-term convergent*, meaning every equation $u = v \in E$ oriented as a rewrite rule $u \to v$ is such that either v is a proper subterm of u or v is a constant. It provides no support for associate and commutative (AC) operations. The AKISS tool, designed to decide reachability and indistinguishability questions on (pairs of) protocols, supports more general equational theories, called optimally reducing [97]. An equational theory $(\mathcal{F}, E)$ in *optimally reducing* if for any $u = v \in E$ oriented as a rewrite rule $u \to v$ and any substitution θ such that all proper subterms of $u\theta$ are in normal form the term $v\theta$ is in normal form. AKISS also provides support for the AC operation XOR. The tool can easily be extended to utilize other tools that solve static equivalence and deduction such as FAST [98] and Maude-NPA [2].

Term rewriting engines. SPAN requires a term rewriting engine to check conditionals and compute normal forms. The tool includes built-in support for term rewriting on convergent equational theories using the unfication algorithm from [99]. For rewriting in the presence of AC operations, support for integration with Maude is also included. SPAN communicates with Maude via asynchronously command line I/O.

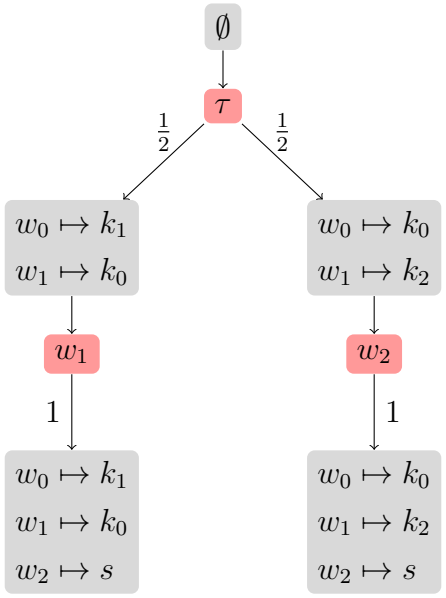
Libraries. To eliminate redundant computation that may occur in different executions paths (branches) of a randomized protocol, SPAN leverages caching enabled by the Guava

library [100] for operations such as rewriting, recipe generation, static equivalence and deduction. SPAN manipulates arbitrary precision numbers provided by Apfloat [101]. Because attacks on randomized protocols are trees (as opposed to sequences), attacks are exported to DOT format, which can be rendered visually using the graphviz framework [102]. For example, consider the protocol with roles

$$\begin{aligned} R_0 &:= [T]\ \mathtt{out}(\tfrac{1}{2} \to k_1, k_0 \# R_1 + \tfrac{1}{2} \to k_0, k_2 \# R_2) \\ R_1 &:= \mathtt{in}(y_1\{k_1\}) \cdot [y_1 = k_1]\ \mathtt{out}(1 \to s) \\ R_2 &:= \mathtt{in}(y_2\{k_2\}) \cdot [y_2 = k_2]\ \mathtt{out}(1 \to s) \end{aligned}$$

where k_0, k_1, k_2 and s are private names. The attack on the protocol deriving s with probability 1 is shown in Figure 4.1. Nodes at even levels of the tree represent the attackers view (a representative from the equivalence class on frames) and nodes at odd levels represent recipes generated by the attacker.

Figure 4.1 SPAN *generated attack tree.*



4.6 EVALUATION

Our experiments were conducted on an Intel core i7 dual quad core processor at 2.67GHz with 12Gb of RAM. The host operating system was 64 bit Ubuntu 16.04.3 LTS. In Section 4.6.1, we evaluate the performance of SPAN on both of the indistinguishability algorithms. Section 4.6.2 evaluates studies the performance of SPAN on safety properties. Our suite of

examples are described informally in Chapter 3. The actual test files are avaliable at [103] for indistinguishability properties and [104] for safety properties.

4.6.1 Evaluation for indistinguishaiblity properties

Our evaluation of SPAN on indistinguishability properties began by examining how the the PFA and OTF algorithms scaled on a variety of examples. The results of the analysis are given in Figure 4.1. For each example, we consider a fixed recipe depth and report the running times for 2 parties as well as the maximum number of parties for which one of the algorithms terminates inside of the timeout bound. On small examples for which the protocols were indistinguishable, we found that the OTF and PFA algorithms were roughly equivalent. On large examples where the protocols were indistinguishable, such as the 3 ballot protocol, the PFA algorithm did not scale as well as the OTF algorithm. In particular, an out-of-memory exception often occurred during construction of the automata or the linear programming constraints. On examples for which the protocols were distinguishable, the OTF algorithm demonstrated a significant advantage. This was a result of the fact that the OTF approach analyzed the model as it was constructed. If at any point during model construction the bisimulation relation was determine not to hold, model construction was halted. By contrast, the PFA algorithm required the entire model to be constructed and stored before any analysis could take place.

Table 4.1 *Expiremental results for indistinguishability properties.* Columns 1 and 2 describe the example being analyzed. Column 3 gives the maximum recipe depth and column 4 indicates when the example protocols were indistinguishable. Columns 5-8 give the running time (in seconds) for the respective algorithms and static equivalence engines. We report OOM for an out of memory exception and TO for a timeout - which occurs if no solution is generated in 60 minutes. Column 9 gives the number of states in the protocol's POMDP and Column 9 gives the number of belief states explored in the OTF algorithm. When information could not be determined due to a failure of the tool to terminate, we report n/a. For protocols using equational theories that were not subterm convergent, we write n/s (not supported) for the KISS engine. All test were conducted using Maude 2.7.1 as the term rewriting engine.

1	2	3	4	5	6	7	8	9	10
PROTOCOL	PARTIES	DEPTH	EQUIV	TIME (S)				STATES	BELIEFS
				PFA		OTF			
				KISS	AISS	KISS	AKISS		
DC-net	2	10	true	n/s	5.5	n/s	4	58	24
DC-net	3	10	true	n/s	OOM	n/s	3013	n/a	286
mix-net	2	10	false	TO	TO	.3	.4	n/a	7
mix-net	5	10	false	OOM	OOM	582	1586	n/a	79654
Evote	2	10	true	1	1	.5	1	34	33
Evote	8	10	true	105	105	131	124	94	93
3 Ballot	2	10	true	n/s	OOM	n/s	1444	n/a	408

In addition to stress-testing the tool, we also examined how each algorithm performed under various parameters of the mix-network example. The results are given in Figure 4.2, where we examine how running times are affected by scaling the number of protocol participants and the recipe depth. Our results coincided with the observations from above. One interesting thing to point out is that the number of beliefs explored on the 5 party example was identical for recipe depth 4 and recipe depth 10. The reason is that, for a given protocol input step, SPAN generates a minimal set of recipes. This is in the sense that if recipes r_0, r_1 are generated at an input step with frame φ, then $r_0\varphi \neq_E r_1\varphi$. For the given number of public names available to the protocol, changing the recipe depth from 4 to 10

did not alter the number of unique terms that could be constructed by the attacker.

Table 4.2 *Detailed experimental results for indistinguishability in mix networks.* The columns have an identical meaning to the ones from Figure 4.1. We report OOM for an out of memory exception and when information could not be determined due to a failure of the tool to terminate, we report n/a.

1	2	3	4	5	6	7	8	9
Parties	Depth	Equiv	Time (s)				States	Beliefs
			PFA		OTF			
			Kiss	Akiss	Kiss	Akiss		
2	1	true	.3	.3	.2	.3	15	12
3	1	true	1	1.2	.4	.9	81	50
4	1	true	47	47	2	6	2075	656
5	1	true	OOM	OOM	34	79	n/a	4032
5	2	false	OOM	OOM	13	33	n/a	1382
5	3	false	OOM	OOM	124	354	n/a	6934
5	4	false	OOM	OOM	580	1578	n/a	79654

4.6.2 Evaluation for state-based safety properties

General performance evaluation Figure 4.3 gives the performance of Span on our benchmark examples. We attempted to verify all protocols with a recipe depth of 10, however, for some examples, Span did not terminate within a reasonable time bound. In such cases, we report the time for a recipe depth of 5. Our analysis revealed that there are two main variables affecting how well our algorithms scale. The first is the number of interleaving possible in a protocol. Observe that the cascade mix, pool mix and and binomial mix all had a high number of beliefs that were explored during analysis. Correspondingly, the analysis time for these examples was quite high in relation to examples like the DC-net and threshold mix, where the number of beliefs explored was much lower. However, observe that the number of beliefs explored in the simple e-vote protocol (6 parties) was drastically lower than the number of beliefs explored in the binomial mix example (3 parties), yet the analysis time for the former was higher. This illustrates the second factor contributing to analysis speed; the average number of original protocol states contained in any belief state. If the number of original protocol states contained in a belief state is high, then the overhead

Table 4.3 *Experimental results for safety properties.* Columns 1-5 describe the example under test, where column 2 is the number of users in the protocol, column 3 is maximum recipe depth, column 4 is the maximum attack probability and column 5 is the security parameter. If the value of column 4 exceeds the value of column 5, then an attack was found. Columns 6 and 7 give the running times (in seconds) under the KISS and Akiss, respectively. Column 8 reports the number of belief states explored during the model checking procedure. All test were conducted using Maude 2.7.1 as the term rewriting engine. For protocols with requiring equational theories with XOR we write n/s (not supported) for the KISS engine.

1	2	3	4	5	6	7	8
PROTOCOL	PARTIES	DEPTH	ATTACK	THRESHOLD	TIME (S)		BELIEFS
					w/ KISS	w/ AKISS	
DC-net	2	10	1/2	1/2	n/s	23	110
Threshold Mix	4	10	1	1/4	22	70	49
Cascade Mix	2	5	1	1/2	917	2832	55303
Pool Mix	3	5	1/2	2/3	1824	6639	26273
Initialized Pool Mix	2	10	1/2	1/2	36	120	714
Binomial Mix	3	5	1/4	1/4	1134	3796	106907
Simple E-Vote	2	10	3/4	3/4	1	3	33
Simple E-Vote	6	10	21/32	21/32	1065	4872	307
FOO 92	2	10	3/4	3/4	321	918	1813
Prêt à Voter	2	10	3/4	7/8	n/s	288	103

associated with computing belief transitions is high. Every transition requires all reachable protocol states to be grouped according to static equivalence. Because KISS and AKISS only determine static equivalence between a pair of states, this grouping can lead to a high number of external queries to the equivalence engines.

Detailed evaluation of threshold mixes In addition to our general performance evaluation of SPAN, we did an in-depth study of how the tool performed for various parameters of the threshold mix-network benchmark. The goal of this study was to determine how the number of parties and recipe depth parameters affected performance. The results are given in Table 4.4. The first four entries of the table analyze the mix-network under depth 1, which corresponds to a passive attacker that can only forward previously received messages or nonces. For passive attackers, there is no attack on mix networks. By contrast, a recipe depth > 2 allows the adversary to forge a valid input to the mix and carry out a flooding attack. SPAN is able to find this attack without having to explore all belief states of the

model. This is the reason that, on the 5 party example, the model checking times for depth 3 are lower than the model checking times for depth 5. On the 5 party example, depth 2 also represents a situation where no attack is possible. This is because our encoding of the mix-network ensures that 1) all inputs during a given round are distinct and 2) only cipher-texts that are encrypted by the mix's public key are accepted. Also, notice again that the model checking times on the 5-party example are the same for depth 4 and 5. This is because increasing the recipe depth from 4 to 5 did not allow any "new" terms to be generated modulo the equational theory.

Table 4.4 *Detailed experimental results for safety properties in mix networks.* Column 1 is the number of users in the protocol and column 2 is maximum recipe depth. Column 3 and 4 gives the running times (in seconds) under the KISS and AKISS engines, respectively. Column 5 gives the number of belief states explored. All test were conducted using the same expiremental setup as in Figure 4.3

1	2	3	4	5
PARTIES	DEPTH	KISS	AKISS	BELIEFS
2	1	.25	.4	10
3	1	1.7	4	16
4	1	19	75	138
5	1	648	3705	1027
5	2	22684	88247	23048
5	3	367	1286	402
5	4	1159	3769	1911
5	5	1167	3705	1911

4.7 CASE STUDY: PRÊT À VOTER

In this section, we demonstrate how the attack on the Prêt à Voter protocol described in Section 3.5 can be modeled in our framework. Let $(\mathcal{F}_{\mathsf{aenc}}, E_{\mathsf{aenc}})$ be the equational theory and sort system from Example 4.2 with the addition of the sort $\mathtt{bit} < \mathtt{msg}$. Our modeling of the protocol will use the theory $(\mathcal{F}_{\mathsf{aenc}} \cup \mathcal{F}_{\mathsf{pv}}, E_{\mathsf{aenc}} \cup E_{\mathsf{pv}})$ where $\mathcal{F}_{\mathsf{pv}}$ is

$$
\begin{aligned}
\mathsf{sign} &:= \mathtt{msg} \times \mathtt{skey} \to \mathtt{msg} \\
\mathsf{getsign} &:= \mathtt{msg} \times \mathtt{pkey} \to \mathtt{bit} \\
\mathsf{chksign} &:= \mathtt{msg} \times \mathtt{pkey} \to \mathtt{msg} \\
\mathsf{hashmod2} &:= \mathtt{msg} \to \mathtt{bit} \\
\mathsf{evenh},\ \mathsf{oddh} &:= \mathtt{msg} \to \mathtt{msg} \\
\mathsf{zero},\ \mathsf{one},\ \mathsf{ok} &:= \to \mathtt{bit} \\
\mathsf{xor} &:= \mathtt{bit} \times \mathtt{bit} \to \mathtt{bit}
\end{aligned}
$$

and E_{pv} contains the following equations.

$$
\begin{aligned}
\mathsf{chksign}(\mathsf{sign}(m, \mathsf{sk}(k)), \mathsf{pk}(k)) &= \mathsf{ok} \\
\mathsf{getsign}(\mathsf{sign}(m, \mathsf{sk}(k)), \mathsf{pk}(k)) &= m \\
\mathsf{hashmod2}(\mathsf{evenh}(m)) &= \mathsf{zero} \\
\mathsf{hashmod2}(\mathsf{oddh}(m)) &= \mathsf{one} \\
\mathsf{xor}(\mathsf{xor}(b', b'), b),\ \mathsf{xor}(b, \mathsf{zero}) &= b \\
\mathsf{xor}(b, b) &= \mathsf{zero}
\end{aligned}
$$

Note that xor is an AC operation and we will henceforth write $\langle , \rangle$ is place of pair. There will be a single teller T, and election authority A and two voters V_1, V_2. The teller will hold two private names k_{t_1} and k_{t_2}, the authority will hold the private name k_a and the voters will hold the names k_{v_1}, k_{v_2}, respectively. We will assume an infinite set $n_1, n_2, \ldots$ of private names to represent nonces and that all of the public keys $\mathsf{pk}(k_{v_1}), \mathsf{pk}(k_{v_2}), \mathsf{pk}(k_{t_1}), \mathsf{pk}(k_{t_2})$ and $\mathsf{pk}(k_a)$ are made available at the beginning of the protocol.

Ballot generation will be modeled by a process that generates a triple (t, g, g') where t is an eligibility token and g, g' are germs that dictate the cyclic offset and onion of a ballot. The eligibility token will be used by the election authority at the voting terminal to verify a voters right to vote. The role modeling the ballot generation process is given in Figure 4.2, where we use the macros $\mathtt{evPair}(g, g') = \langle(\mathsf{evenh}(g), \mathsf{evenh}(g')\rangle$ and $\mathtt{oddPair}(g, g') = \langle(\mathsf{evenh}(g), \mathsf{oddh}(g')\rangle$. Because there is no guarantee that the sum of hashes of the germs modulo 2 is evenly distributed, we select an non-uniform distribution for modeling purposes.

Figure 4.2 *Role modeling Prêt à Voter ballot generation.*

$$
\begin{aligned}
[\top]\ \mathtt{out}(&3/4 \to \mathsf{aenc}(\langle t_1, \mathtt{evPair}(g_0, g_1)), n_1, \mathsf{pk}(k_{v_1})) + \\
&1/4 \to \mathsf{aenc}(\langle t_1, \mathtt{oddPair}(g_0, g_1)), n_1, \mathsf{pk}(k_{v_1}))) \\
[\top]\ \mathtt{out}(&3/4 \to \mathsf{aenc}(\langle t_2, \mathtt{evPair}(g'_0, g'_1)), n_2, \mathsf{pk}(k_{v_2})) + \\
&1/4 \to \mathsf{aenc}(\langle t_2, \mathtt{oddPair}(g'_0, g'_1)), n_2, \mathsf{pk}(k_{v_2})))
\end{aligned}
$$

In the second part of the protocol, each voter receives his/her eligibility token and germs, authenticates with the election authority at the voting both and then sends vote (the XOR of

the cyclic offset and the vote position) and the onion to the teller. Messages received by the teller must be signed by the election authority. The role modeling this process for voter V_i ($i \in \{1,2\}$) is given in Figure 4.3 where we use the macros $\mathtt{germs}_i = (\mathsf{adec}(y_i, \mathsf{sk}(k_{v_i})))$, $\mathtt{germ}_i^0 = (\mathsf{snd}(\mathsf{snd}(\mathtt{germs}_i)))$, $\mathtt{germ}_i^1 = (\mathsf{fst}(\mathsf{snd}(\mathtt{germs}_i)))$, $\mathtt{off}_i = \mathsf{xor}(\mathtt{germ}_i^0, \mathtt{germ}_i^1)$ and $\mathtt{onion}_i = \mathsf{aenc}(\langle \mathtt{germ}_i^0, \mathsf{aenc}(\langle \mathtt{germ}_i^1, D_i\rangle, n_3, \mathsf{pk}(k_{t_1}))\rangle, n_4, \mathsf{pk}(k_{t_2}))$. The values D_0, D_1 are assumed to be nonces and n_3, n_4 should be distinct for each voter. For modeling purposes, we assume that each voter selects one of the two candidates with probability $\frac{1}{2}$. The vote itself as represented as a position, either zero or one, that is combined with the offset to identify the chosen candidate.

Figure 4.3 *Role modeling voting terminal with voter V_i.*

$\mathtt{in}(y_i\{\mathsf{aenc}(\langle t_i, \mathtt{msg}, \mathtt{nonce}, \mathsf{pk}(k_{v_i})\rangle)\})$
$[\mathsf{fst}(\mathsf{ballot}_i) == t_i]$
$\mathtt{out}(1/2 \to \mathsf{sign}(\langle \mathsf{xor}(\mathtt{off}_i, \mathsf{zero}), \mathtt{onion}_i\rangle, \mathsf{sk}(k_a)) +$
$\quad 1/2 \to \mathsf{sign}(\langle \mathsf{xor}(\mathtt{off}_i, \mathsf{one}), \mathtt{onion}_i\rangle, \mathsf{sk}(k_a)))$

In the final phase of the protocol, the teller collects both of the ballot forms, reconstructs the votes from the onion and the vote position and then publishes the votes in random order. This process is modeled using the role from Figure 4.4 where we use the macros $\mathtt{ballot_j} = \mathsf{getsign}(y_j, \mathsf{pk}(k_a))$ (for $j \in \{3,4\}$), $\mathtt{vpos}_j = \mathsf{fst}(\mathtt{ballot}_j)$,

$$\mathtt{recgrm}_j^0 = \mathsf{hashmod2}(\mathsf{fst}(\mathsf{adec}(\mathsf{snd}(\mathtt{ballot_j}), \mathsf{sk}(k_{t_2}))))$$

and

$$\mathtt{recgrm}_\mathtt{j}^1 = \mathsf{hashmod2}(\mathsf{fst}(\mathsf{adec}(\mathsf{snd}(\mathsf{adec}(\mathsf{snd}(\mathtt{ballot_j}), \mathsf{sk}(k_{t_2}))), \mathsf{sk}(k_{t_1})))).$$

Figure 4.4 *Role modeling teller.*

$\mathtt{in}(y_3\{\mathsf{sign}(\mathtt{msg}, \mathsf{pk}(k_a))\})$
$\mathtt{in}(y_4\{\mathsf{sign}(\mathtt{msg}, \mathsf{pk}(k_a))\})$
$[\mathsf{chksign}(y_3, \mathsf{pk}(k_a)) = \mathsf{ok} \wedge$
$\mathsf{chksign}(y_3, \mathsf{pk}(k_a)) = \mathsf{ok} \wedge y_3 \neq y_4]$
$\mathtt{out}(1/2 \to \mathsf{xor}(\mathtt{vpos}_3, \mathsf{xor}(\mathtt{recgrm}_3^0, \mathtt{recgrm}_3^1)),$
$\qquad \mathsf{xor}(\mathtt{vpos}_4, \mathsf{xor}(\mathtt{recgrm}_4^0, \mathtt{recgrm}_4^1)) +$
$\mathtt{out}(1/2 \to \mathsf{xor}(\mathtt{vpos}_4, \mathsf{xor}(\mathtt{recgrm}_4^0, \mathtt{recgrm}_4^1)),$
$\qquad \mathsf{xor}(\mathtt{vpos}_3, \mathsf{xor}(\mathtt{recgrm}_3^0, \mathtt{recgrm}_3^1)))$

There are four possible outcomes of this simple election, two in which both parties vote

for the same candidate, one in which V_1 votes for party 0 and V_2 votes for party 1 and one in which the votes from the preceding scenario are switched. Because we have modeled each voter selecting one of the candidates with probability $\frac{1}{2}$, each of these outcomes happens with probability $\frac{1}{4}$. When both parties vote for the same candidate the adversary can guess who V_1 (resp. V_2) voted for with probability one. When their votes don't match, the adversary should only be able to guess which candidate each voter selected. This means, if the protocol is secure, the adversary should not be able to guess a voters selection with probability $> \frac{3}{4}$. This security requirement can be enforced by adding a role of the form

$$\mathtt{in}(y_5\{\mathtt{bit}\}) \cdot [y_5 = \mathtt{vote}_i]\ \mathtt{out}(1 \to s_i)$$

to the end of the protocol, where $\mathtt{vote}_i$ represents the vote of V_i. Because the vote is not stored in any local variable, a role invocation can be added to the voting phase of the protocol given in Figure 4.3. In the case where V_i votes for the first party, a role with $\mathtt{vote}_i = \mathsf{zero}$ is invoked. In the other case, a role with $\mathtt{vote}_i = \mathsf{one}$ is invoked. Let P be protocol modeling Prêt à Voter described above. The protocol satisfies vote privacy if $P \models_{E_{\mathsf{aenc}} \cup E_{\mathsf{pv}}, \frac{1}{4}} \mathsf{secret}(\{s_i\})$ for $i \in \{0, 1\}$.

CHAPTER 5: COMPOSITION OF INDISTINGUISHABILITY PROPERTIES

In this chapter, we study the problem of composition for randomized security protocols when the protocols are allowed to share data, such as keys. We begin our study by considering single-session protocols. In this setup, we give a set of conditions under which the composition of indistinguishable protocols under disjoint equational theories preserves indistinguishability (Theorem 5.1 on page 53). More precisely, consider indistinguishable protocols P and Q over equational theory E_a, and indistinguishable protocols P' and Q' over equational theory E_b, where E_a and E_b are disjoint. We show that the composition of P and P' is indistinguishable from the composition of Q and Q', provided the shared secrets between P and P' and those between Q and Q' are kept with probability 1. While such a result also holds for non-randomized protocols (see [54, 56] for example), randomization presents its own challenges.

The first challenge arises from the fact that even if P and Q (resp. P' and Q') do not leak shared secrets, they may still reveal equalities that hold amongst the shared secrets. Revealing these equalities may, in some cases, allow the attacker to infer the result of a private coin toss (See Example 5.3 on page 54). Consequently, our composition theorem requires that P and Q remain indistinguishable even when such equalities are revealed. The revelation of the equalities is achieved by adding actions to protocols P and Q that reveal "hashes" of shared secrets.

As in non-randomized protocols [54, 56], the proof proceeds by showing that it suffices to reduce the problem to the case when P (Q) does not share any secrets with P' (Q' respectively). In the non-randomized setting, this is achieved by observing that if the composition of P and P' is not indistinguishable from the composition of Q and Q', then there must be a trace $\overline{o}$ and an individual execution ρ in the composition of P and P' (or of Q and Q') such that ρ has trace $\overline{o}$ and there is no execution in the composition of Q and Q' (or of P and P' respectively) with the trace $\overline{o}$. It is then observed that if the shared secrets between P and P', and between Q and Q', are *re-initialized* to fresh values respecting the same equalities amongst them as in the execution ρ, then the transformed protocols continue to remain distinguishable. After this re-initialization of the shared values, the protocols no longer share data. For randomized protocols, we no longer have an individual execution that witnesses protocol distinguishability. Instead we have an attacker $\mathcal{A}$ and a trace $\overline{o}$ which occurs with different probabilities when the protocols interact with $\mathcal{A}$. Observe that the executions corresponding to the trace $\overline{o}$ will form a tree, and different equalities amongst the shared secrets may hold in different branches. Thus, a simple transformation as in the case

of non-randomized protocols no longer suffices (see Example 5.2 on page 54). Instead, we have to perform a non-trivial inductive argument (with induction on number of coin tosses) to show that it suffices to consider the case when P (resp. Q) does not share any secrets with P' (resp. Q').

Our second result concerns the case when the equational theories E_a and E_b are the same, each containing cryptographic primitives for symmetric encryption, symmetric decryption and hashes (see Theorem 5.2 on page 72). For this case, we show that the composition of randomized protocols preserves indistinguishablility when the protocols are allowed to share secrets, provided protocol messages are tagged with the information of which protocol they belong to. As in the case of non-randomized protocols, this is achieved by showing that in presence of tagging, the protocols can be transformed to new protocols $P_{\mathsf{new}}, P'_{\mathsf{new}}, Q_{\mathsf{new}}, Q'_{\mathsf{new}}$ such that P_{new} and Q_{new} are indistinguishable protocols with equational theory E_{new}, and P'_{new} and Q'_{new} are indistinguishable protocols with disjoint equational theory E'_{new}. Thus, this result follows from the first result of this chapter.

Our final result in this chapter extends the above result to the case of unbounded number of sessions (see Theorem 5.3 on page 88). We again consider the case when the equational theories E_a and E_b are the same, containing cryptographic primitives for symmetric encryption, symmetric decryption and hashes. In order to achieve this result, we additionally require that messages from each session are tagged with a unique session identifier.

5.1 COMPOSITION FRAMEWORK

5.1.1 Contexts

In this section we introduce contexts, our vehicle for protocol composition. A context, is like a process except that it contains "holes". These holes can be substituted by roles, creating a composed process. To define contexts, we will assume a countable set of process variables $\mathcal{X}_c$, whose elements will be denoted by $\Box, \Box_1, \ldots, \Box_m$. Contexts are defined in Figure 5.1, where we begin by defining basic contexts, which extend roles by allowing for the occurence of a single process variable from $\mathcal{X}_c$ in any branch of the role. Basic contexts will be denoted by $D[\Box], D_1[\Box], D_2[\Box], \ldots, D_n[\Box]$. For a basic context $D[\Box]$ and a role R, $D[R]$ denotes the process that results from replacing every occurrence of $\Box$ in D by R. Using basic contexts, we can define contexts as the parallel composition of a set of basic contexts preceded by a sequential composition of fresh variable creations and variable assignments. The prefix of variable creations and assignments is used to instantiate data common to one or more basic contexts. A process is nothing but a context that does not contain any process

variables. We will use $C, C_1, C_2, \ldots, C_n$ to denote contexts. For a context $C[\Box_1, \ldots, \Box_m]$ and roles $R_1, \ldots, R_m$, $C[B_1, \ldots, B_m]$ denotes the process that results from replacing each process variable $\Box_i$ by R_i.

Figure 5.1 *Context sytax.*

Basic Contexts

$$D[\Box] ::= \Box \,\big|\, R \,\big|\, D[\Box] \cdot R \,\big|\, R \cdot D[\Box] \,\big|\, D[\Box] +_p^{\ell} D[\Box]$$

Contexts $[a_i \in \{\nu x, (x := u)\}]$

$$C[\Box_1, \ldots, \Box_m] ::= a_1^{\ell_1} \cdot \ldots \cdot a_n^{\ell_n} \cdot (D_1[\Box_1] \mid \ldots \mid D_m[\Box_m])$$

As we do for processes, we will drop labels from a context when they are not relevant. A context $C[\Box_1, \ldots, \Box_m] = a_1 \cdot \ldots \cdot a_n \cdot (D_1[\Box_1] \mid \ldots \mid D_m[\Box_m])$ is said to be well-formed if every operator has a unique label and for any labels ℓ_1 and ℓ_2 occurring in D_i and D_j for $i, j \in \{1, 2, \ldots, m\}$, $i \neq j$ iff $[\ell_1] \neq [\ell_2]$. For the remainder of this work, contexts are assumed to be well-formed. A process that results from replacing process variables in a context by roles is also assumed to be well-formed. Whenever new actions are added to a process, their labels are assumed to be fresh and not equivalent to any existing labels of that process. In the following example, we demonstrate how contexts can be used to instantiate data (keys) to be used by roles that are later substituted into the context.

Example 5.1 *Let* $(\mathcal{F}_{\mathsf{dh}}, E_{\mathsf{dh}})$ *the equational theory with the signature* $\mathcal{F}_{\mathsf{dh}} = \{\mathsf{f}/1, \mathsf{g}/1, \mathsf{mac}/1\}$ *and the equations* $E_{\mathsf{dh}} = \{f(g(y), x) = f(g(x), y)\}$ *modeling the Diffie-Hellman primitives, i.e.* $f(x, y) = x^y \bmod p$, $g(y) = \alpha^y \bmod p$ *for some group generator* α. *We use* mac *for a keyed hash function. Define* $C[\Box_1, \Box_2] = \nu k_h \cdot D_1[\Box_1] \mid D_2[\Box_2]$ *to be the context that models (a variant of) the Diffie-Hellman protocol where* D_1 *and* D_2 *are below.*

$$\begin{aligned} D_1 &= \nu x \cdot \mathtt{out}(\mathsf{g}(x)) \cdot \mathtt{out}(\mathsf{mac}(\mathsf{g}(x), \mathsf{a}, k_h)) \cdot \mathtt{in}(z) \cdot \\ &\quad \mathtt{in}(z') \cdot [z' = \mathsf{mac}(z, \mathsf{b}, k_h)] \cdot (k_1 := f(x, z)) \cdot \Box_1 \\ D_2 &= \nu y \cdot \mathtt{out}(\mathsf{g}(y)) \cdot \mathtt{out}(\mathsf{mac}(\mathsf{g}(y), \mathsf{b}, k_h)) \cdot \mathtt{in}(z) \cdot \\ &\quad \mathtt{in}(z') \cdot [z' = \mathsf{mac}(z, \mathsf{a}, k_h)] \cdot (k_2 := f(y, z)) \cdot \Box_2 \end{aligned}$$

5.1.2 Composition under disjoint data

In the case of non-randomized protocols, it is well known that composition preserves indistinguishability when protocols do not share data. Recall that we have introduced a new notion of indistinguishability for randomized protocols wherein two protocols P and

Q are equivalent if, for every attacker $\mathcal{A}$ and trace $\overline{o}$, the event $\overline{o}$ has equal probability in the Markov chains $[\![P^{\mathcal{A}}]\!]$ and $[\![Q^{\mathcal{A}}]\!]$. A cornerstone of our result establishes that parallel composition is a congruence with respect to indistinguishability when protocols do not share data.

Lemma 5.1 *Let P, P', Q and Q' be closed processes such that $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \emptyset$ and $\mathsf{vars}(P') \cap \mathsf{vars}(Q') = \emptyset$. If $P \approx P'$ and $Q \approx Q'$ then $P \mid Q \approx P' \mid Q'$.*

In the absence of probabilistic choice, Lemma 5.1 is obtained by transforming an attacker $\mathcal{A}$ for $P \mid Q$ into an attacker $\mathcal{A}'$ that "simulates" Q to P (and vice versa). When a term created by Q is forwarded to P by $\mathcal{A}$, the attacker $\mathcal{A}'$ forwards a new recipe in which the nonces in the term created by Q are replaced by fresh nonces created by the adversary. This technique is not directly applicable in the presence of randomness, where the adversary must forward a common term to P in every indistinguishable execution of $P \mid Q$. For example, if the outputs n_1 and $\mathsf{h}(n_2)$ from Q are forwarded by $\mathcal{A}$ to P in two indistinguishable executions, the recipe constructed by $\mathcal{A}'$ must be identical for both executions. Further complicating matters, if n_2 is later revealed by Q, the simulation of n_1 and $\mathsf{h}(n_2)$ to P via a common term can introduce in-equivalences that were not present in the original executions. We prove the result by showing $P \approx P' \Rightarrow P \mid Q \approx P' \mid Q$ and $Q \approx Q' \Rightarrow P \mid Q \approx P \mid Q'$, which together imply Lemma 5.1. Each sub-goal is obtained by first inducting on the number of probabilistic choices in the common process. In the case of $P \mid Q \approx P' \mid Q$, for example, this allows one to formulate an adversary $\mathcal{A}$ for $P \mid Q$ (resp. $P' \mid Q$) as a combination of two disjoint adversaries. We then appeal to a result on POMDPs where we prove that the asynchronous product of POMDPs preserves indistinguishability. The full proof of Lemma 5.1 is given in Section 5.4.

5.1.3 Composition under disjoint primitives

In our composition results, we would like to argue that if two contexts $C[\Box]$ and $C'[\Box]$ are "equivalent" and two basic process B and B' are "equivalent", then the processes $C[B]$ and $C'[B']$ are indistinguishable. In such a setup, contexts can instantiate data (keys) that are used by and occur free in the basic processes. This setup provides a natural way to model and reason about protocols that begin by carrying out a key exchange before transitioning into another phase of the protocol. It is worth pointing out that the combination of key exchange with anonymity protocols can indeed lead to errors. For example, it was observed in [105] that the RSA implementation of mix networks leads to a degradation in the level of

anonymity provided by the mix. The formalization of our main composition result for this chapter is as follows.

Theorem 5.1 *Let* $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot (D_1[\Box_1] \mid \ldots \mid D_n[\Box_n])$ *(resp.* $C'[\Box_1, \ldots, \Box_n] = \nu k'_1 \cdot \ldots \cdot \nu k'_m \cdot (D'_1[\Box_1] \mid \ldots \mid D'_n[\Box_n])$*) be a context over* $\mathcal{F}_c$ *with labels from* $\mathcal{L}_c$*. Further let* $B_1, \ldots, B_n$ *(resp.* $B'_1, \ldots, B'_n$*) be roles over* $\mathcal{F}_b$ *with labels from* $\mathcal{L}_b$*. For* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$ *and* $\sharp \notin \mathcal{F}_b \cup \mathcal{F}_c$*, assume that the following hold.*

1. $\mathsf{fv}(C) = \mathsf{fv}(C') = \emptyset$, $\mathsf{fv}(B_i) = \{x_i\}$ *and* $\mathsf{fv}(B'_i) = \{x'_i\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *and* $\mathsf{vars}(C') \cap \mathsf{vars}(B'_i) = \{x'_i\}$
3. $C[B_1, \ldots, B_n]$ *and* $C'[B'_1, \ldots, B'_n]$ *are ground*
4. $C[B_1, \ldots, B_n] \models_{E,1} \mathsf{secret}(x_1, \ldots, x_n)$ *and* $C'[B'_1, \ldots, B'_n] \models_{E,1} \mathsf{secret}(x'_1, \ldots, x'_n)$
5. $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \approx C'[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}]$
6. $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \approx \nu k \cdot (x'_1 := k) \cdot \ldots \cdot (x'_n := k) \cdot (B'_1 \mid \ldots \mid B'_n)$

Then $C[B_1, \ldots, B_n] \approx C'[B'_1, \ldots, B'_n]$.

The result is achieved by showing that if $C[B_1, \ldots, B_n]$ is not distinguishable from $C'[B'_1, \ldots, B'_n]$ then one of conditions 5 or 6 from Theorem 5.1 is violated. More specifically, we use an offending trace $\overline{o}$ under an attacker $\mathcal{A}$ for $C[B_1, \ldots, B_n] \not\approx C'[B'_1, \ldots, B'_n]$, i.e. a trace such that $\mathsf{prob}_{C[B_1,\ldots,B_n]}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{C'[B'_1,\ldots,B'_n]}(\overline{o}, \mathcal{A})$, to construct a trace $\overline{o}_0$ that witnesses a violation of condition 5 or 6 from Theorem 5.1. In particular, we can show that if $C[B_1, \ldots, B_n] \not\approx C'[B'_1, \ldots, B'_n]$ then

$$\begin{gathered} C[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \mid B_0 \cdot (B_1 \mid \ldots \mid B_n) \\ \not\approx \\ C'[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \mid B'_0 \cdot (B'_1 \mid \ldots \mid B'_n) \end{gathered} \tag{5.1}$$

where B_0 and B'_0 are processes that bind $\{x_1, \ldots, x_n\}$ and $\{x'_1, \ldots, x'_n\}$, respectively. This transformation is a non-trival extension of a result from [54, 57] which allows a process $P \mid Q$, where P and Q share common variables but are over disjoint signatures, to be transformed into an "equivalent" process $P' \mid Q'$ where variables are no longer shared. Variables of Q are re-initialized in Q' according to the equational equivalences they respect in an execution of $P \mid Q$. Unlike nondeterministic processes, where executions are sequences, executions in randomized processes form a tree where variables can receive different values in

different branches of the tree. From equation 5.1, we can apply Lemma 5.1 to achieve either $C[\mathtt{out}(\sharp(x_1)),\ldots,\mathtt{out}(\sharp(x_n))] \not\approx C'[\mathtt{out}(\sharp(x_1)),\ldots,\mathtt{out}(\sharp(x_n))]$ or $B_0 \cdot (B_1 \mid \ldots \mid B_n) \not\approx B_0' \cdot (B_1' \mid \ldots \mid B_n')$. In the former case, we have contradicted condition 5 of Theorem 5.1. If we achieve $B_0 \cdot (B_1 \mid \ldots \mid B_n) \not\approx B_0' \cdot (B_1' \mid \ldots \mid B_n')$, we additionally need to transform an adversary that witnesses this distinguishablility to an adversary that witnesses the distinguishablility $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \not\approx \nu k \cdot (x_1' := k) \cdot \ldots \cdot (x_n' := k) \cdot (B_1' \mid \ldots \mid B_n')$. The presence of randomness makes this transformation tricky, as illustrated by Example 5.2 below.

Example 5.2 *Define* $B_0, B_0' = \nu k_1 \cdot \nu k_2 \cdot (x_1 := k_1) \cdot (x_2 := k_2)$, $B_1, B_1' = \mathtt{out}(\mathsf{h}(x_1))$, $B_2 = \mathtt{in}(y) \cdot (\mathtt{out}(y) +_{\frac{1}{2}} \mathtt{out}(\mathsf{h}(x_2)))$ *and* $B_2' = \mathtt{in}(y) \cdot (\mathtt{out}(\mathsf{h}(x_2)) +_{\frac{1}{2}} \mathtt{out}(\mathsf{h}(x_2)))$. *Consider the adversary* $\mathcal{A}$ *for* $B_0 \cdot (B_1 \mid B_2)$ *(resp.* $B_0' \cdot (B_1' \mid B_2')$*) that forwards the output of* B_1 *(resp.* B_1'*) to* B_2 *(resp.* B_2'*).* $\mathcal{A}$ *is a witness to the distinguishability of* $B_0 \cdot (B_1 \mid B_2)$ *and* $B_0' \cdot (B_1' \mid B_2')$*, but it does not witness the distinguishability of* $\nu k_1 \cdot (x_1 := k_1) \cdot (x_2 := k_1) \cdot (B_1 \mid B_2)$ *and* $\nu k_1 \cdot (x_1 := k_1) \cdot (x_2 := k_1) \cdot (B_1' \mid B_2')$*. We can, however, transform the attacker* $\mathcal{A}$ *to an attacker* $\mathcal{A}'$ *that witnesses* $\nu k_1 \cdot (x_1 := k_1) \cdot (x_2 := k_1) \cdot (B_1 \mid B_2) \not\approx \nu k_1 \cdot (x_1 := k_1) \cdot (x_2 := k_1) \cdot (B_1 \mid B_2')$*. The details of this transformation, as well as the full proof of Theorem 5.1 can be found in Section 5.5*

5.1.4 Difficulties with composition

In the setup from Theorem 5.1, observe that $C[\Box], C'[\Box]$ contain process (free) variables. As a result, trace equivalence cannot directly be used to equate these objects. One natural notion of equivalence between $C[\Box]$ and $C'[\Box]$ is achieved by requiring $C[B_0] \approx C'[B_0]$ under all assignments of $\Box$ to a basic process B_0. While mathematically sufficient for achieving composition, such a definition creates a non-trivial computational overhead. Instead, our result is able to guarantee safe composition when $C[B_0] \approx C'[B_0]$ for a *single* instantiation of B_0. A natural selection for B_0 is the empty process $[\top]$. We illustrate why such a choice is insufficient in Example 5.3.

Example 5.3 *Consider the contexts defined below.*

$$\begin{aligned} C[\Box_1, \Box_2] &= \nu k_1 \cdot \nu k_2 \cdot ((x_1 := k_1) \cdot \Box_1 \mid (x_2 := k_1) \cdot \Box_2 +_{\frac{1}{2}} (x_2 := k_2) \cdot \Box_2) \\ C'[\Box_1, \Box_2] &= \nu k_1 \cdot \nu k_2 \cdot ((x_1 := k_1) \cdot \Box_1 \mid (x_2 := k_1) \cdot \Box_2 +_{\frac{1}{2}} (x_2 := k_1) \cdot \Box_2). \end{aligned}$$

Notice that C *and* C' *differ in that* C *assigns* x_2 *to* k_1 *or* k_2*, each with probability* $\frac{1}{2}$*, while* C' *assigns* x_2 *to* k_1 *with probability* 1*. For the basic processes* $B_1 = \mathtt{out}(\mathsf{h}(x_1))$ *and*

$B_2 = \mathtt{out}(\mathsf{h}(x_2))$. *We have* $C[[\top],[\top]] \approx C'[[\top],[\top]]$ *as for any adversary* $\mathcal{A}$, $C[[\top],[\top]]^{\mathcal{A}}$ *and* $C'[[\top],[\top]]^{\mathcal{A}}$ *yield a single common trace that occurs with probability* 1. *On the other hand,* $C[B_1, B_2] \not\approx C'[B_1, B_2]$. *This is because there is an adversary* $\mathcal{A}'$ *for the processes* $C[B_1, B_2]$ *and* $C'[B_1, B_2]$ *such that the trace outputting* $\mathsf{h}(k_1), \mathsf{h}(k_1)$ *occurs with probability* 1 *in* $C'[B_1, B_2]^{\mathcal{A}'}$ *and with probability* $\frac{1}{2}$ *in* $C[B_1, B_2]^{\mathcal{A}'}$. *The second trace of* $C[B_1, B_2]^{\mathcal{A}'}$ *outputs* $\mathsf{h}(k_1), \mathsf{h}(k_2)$ *with probability* $\frac{1}{2}$.

The problematic behavior arising in Example 5.3 occurs when basic processes reveal equalities among the shared secrets from the context. Revealing these equalities may, in some cases, allow the attacker to infer the result of a private coin toss. Consequently, our composition theorem must require contexts to remain secure even when such equalities are revealed. As was the case with composition contexts, our result also relies on a notion of indistinguishablilty between basic processes B and B' that contain free variables. Universal quantification over the free variables results in a non-trivial computational overhead. However, we are able to show that when B is distinguishable from B' under some instantiation of the free variables, then B and B' can also be shown to be distinguishable when all of the free variables take the same value. This allows us to prove a stronger result by requiring a weaker condition on the equivalence between B and B'. Another subtle component of Theorem 5.1 is condition 1, which allows each basic process to share only a single variable with the context. As demonstrated by Example 5.4, the composition theorem does not hold when this restriction is relaxed.

Example 5.4 *Consider the context and processes below.*

$$\begin{aligned} C[\Box] &= \nu k_1 \cdot \nu k_2 \cdot \nu k_3 \cdot (x_1 := k_1) \cdot (x_2 := k_2) \cdot (x_3 := k_3) \cdot \Box \\ B_1 &= \mathtt{out}(\mathsf{senc}(x_1, x_3)) \cdot \mathtt{out}(\mathsf{senc}(x_1, x_3)) \\ B_2 &= \mathtt{out}(\mathsf{senc}(x_1, x_3)) \cdot \mathtt{out}(\mathsf{senc}(x_1, x_2)). \end{aligned}$$

For $B_0 = \nu k \cdot (x_1 := k) \cdot (x_2 := k) \cdot (x_3 := k)$ *we have* $B_0 \cdot B_1 \approx B_0 \cdot B_2$ *but* $C[B_1] \not\approx C[B_2]$. *Indeed, observe that* $B_0 \cdot B_1$ *and* $B_0 \cdot B_2$ *have a single trace that outputs* $\mathsf{senc}(k, k), \mathsf{senc}(k, k)$. *However, an adversary that executes* $C[B_1], C[B_2]$ *to completion produces a trace that outputs* $\mathsf{senc}(k_1, k_3), \mathsf{senc}(k_1, k_3)$ *for* $C[B_1]$ *and a trace that outputs* $\mathsf{senc}(k_1, k_3), \mathsf{senc}(k_1, k_2)$ *for* $C[B_2]$.

In Theorem 5.1, condition 4 requires that the composed processes do not reveal the shared secrets with probability 1. One might also be interested in the quantitative version of this result, where the probability of revealing the shared secrets is below a given threshold. Unfortunately, condition 4 cannot be relaxed, as shown by Example 5.5 below.

Example 5.5 *Consider the contexts*

$$\begin{aligned} C[\Box] &= \nu k_1 \cdot (x_1 := k_1) \cdot \Box \cdot \mathtt{in}(y) \cdot \nu k_2 \cdot [y = x_1] \cdot \mathtt{out}(ok) \\ C'[\Box] &= \nu k_1 \cdot (x_1 := k_1) \cdot \Box \cdot \mathtt{in}(y) \cdot \nu k_2 \cdot [y = k_2] \cdot \mathtt{out}(ok) \end{aligned}$$

and the basic process $B = \nu n \cdot \mathtt{out}(x_1) +_{\frac{1}{2}} \nu n \cdot \mathtt{out}(n)$. *Observe that* $C[B] \models_{E,\frac{1}{2}} \mathsf{secret}(x_1)$ *and* $C'[B] \models_{E,\frac{1}{2}} \mathsf{secret}(x_1)$. *Furthermore,* $C[\mathtt{out}(\#(x_1))] \approx C'[\mathtt{out}(\#(x_1))]$ *and* $\nu k \cdot (x_1 := k) \cdot B \approx \nu k \cdot (x_1 := k) \cdot B$. *However,* $C[B] \not\approx C'[B]$.

5.1.5 Application of the composition framework

We demonstrate an application of Theorem 5.1 using the DC-net protocol from Section 3.2.

Example 5.6 *We model the DC-net protocol in our formalism as follows. We will use the equational theory* $(\mathcal{F}_{\mathsf{dc}}, E_{\mathsf{dc}})$ *with signature* $\mathcal{F}_{\mathsf{dc}} = \{0, 1, \oplus, \mathsf{senc}, \mathsf{sdec}, \mathsf{pair}, \mathsf{fst}, \mathsf{snd}, \mathsf{val}\}$ *and the equations* E_{dc} *given below.*

$$\begin{aligned} \mathsf{sdec}(\mathsf{senc}(m,k),k) &= m \\ \mathsf{fst}(\mathsf{pair}(x,y)) &= x \\ \mathsf{snd}(\mathsf{pair}(x,y)) &= y \\ (x \oplus y) \oplus z &= x \oplus (y \oplus z) \\ x \oplus 0 &= x \\ x \oplus x &= 0 \\ x \oplus y &= y \oplus x \\ \mathsf{val}(m,0,b_1,b_2) &= \mathsf{pair}(b_1 \oplus m, b_2) \\ \mathsf{val}(m,1,b_1,b_2) &= \mathsf{pair}(b_1, b_2 \oplus m) \end{aligned}$$

The roles of Alice and Bob in this protocol are defined in our process syntax as follows.

$$\begin{aligned} A &= ((b_0 := 0) +_{\frac{1}{2}} (b_0 := 1)) \cdot ((b_1 := 0) +_{\frac{1}{2}} (b_1 := 1)) \cdot ((b_2 := 0) +_{\frac{1}{2}} (b_2 := 1) \cdot \\ &\quad \mathtt{out}(\mathsf{senc}(\mathsf{pair}(b_0, \mathsf{pair}(b_1, b_2)), k_1) \cdot \mathtt{out}(\mathsf{val}(m_A, b_0, b_1, b_2)) \\ B &= \mathtt{in}(z) \cdot (y := \mathsf{sdec}(z, k_2)) \cdot (b_0 := \mathsf{fst}(y)) \cdot (b_1 := \mathsf{fst}(\mathsf{snd}(y))) \cdot \\ &\quad (b_2 := \mathsf{snd}(\mathsf{snd}(y))) \cdot \mathtt{out}(\mathsf{val}(m_B, b_0 \oplus 1, b_1, b_2)) \end{aligned}$$

Notice that the output of Bob depends on the value of Alice's coin flip. Because the process calculus of our composition framework does not contain else branches, the required functionality is embedded in the equational theory. Also notice that the communication between Alice and Bob in the above specification requires pre-established secret keys k_1 *and* k_2 *that can be generated by first running a key-exchange protocol. Let* $C[\Box_1, \Box_2]$ *be the context modeling*

the Diffie-Hellman protocol from Example 5.1. If Alice holds the bit b and Bob holds the bit b′, the entire protocol can be modeled as $C[(m_A := b) \cdot A, (m_B := b') \cdot B]$.

The security property of DC-nets can be formalized as follows.

Example 5.7 *Consider the DC-net protocol defined in Example 5.6 which is designed to ensure that an observer of the protocol's output can obtain Alice and Bob's bits but cannot distinguish the party to which each bit belongs. This property can be modeled by the indistinguishablity* $C[(m_A := 0) \cdot A \mid (m_B := 1) \cdot B] \approx C[(m_A := 1) \cdot A \mid (m_B := 0) \cdot B]$ *which says that any attacker for the DC-net protocol will observe identical probabilities for every sequence of protocol outputs, regardless of the bits that Alice and Bob hold in their messages.*

Using Theorem 5.1, we can verify the DC-net protocol in a modular way.

Example 5.8 *Consider the indistinguishablity property from Example 5.7. Using the results established in Theorem 5.1, the verification effort for the property can be reduced to verifying the following set of simpler properties, where* $K = \nu k \cdot (k_1 := k) \cdot (k_2 := k)$.

1. $C[B_1, \ldots, B_n] \models_{E,1} \mathsf{secret}(x_1, \ldots, x_n)$ *and* $C'[B'_1, \ldots, B'_n] \models_{E,1} \mathsf{secret}(x'_1, \ldots, x'_n)$
2. $K \cdot ((m_a := 0) \cdot A \mid (m_b := 1) \cdot B) \approx K \cdot ((m_a := 1) \cdot A \mid (m_b := 0) \cdot B)$

Property 1 can also be verified modularly using Theorem 6.1 from Section 6.1. When the contexts in the equivalence are not the same, one must also verify property 5 from Theorem 5.1.

5.2 COMBINING INTRUDER KNOWLEDGE IN DISJOINT EQUATIONAL THEORIES

In this section, let $e \in \{a, b, c\}$, $d \in \{b, c\}$ and $\overline{d}$ denote $\{b, c\} \setminus d$. Let $(\mathcal{F}_a, E_a)$, $(\mathcal{F}_b, E_b)$ and and $(\mathcal{F}_c, E_c)$ be disjoint non-trivial equational theories. We will assume that $\mathcal{F}_e$ has private names $\mathcal{N}^e_{\mathsf{priv}}$ and public names $\mathcal{N}^e_{\mathsf{pub}}$. Further, we will assume that $E_a = \emptyset$ and $\mathcal{F}_a = \mathcal{N}^a_{\mathsf{priv}} \cup \mathcal{N}^a_{\mathsf{pub}}$. Define $(\mathcal{F}, E)$ as the equational theory with signature $\mathcal{F} = \mathcal{F}_a \cup \mathcal{F}_b \cup \mathcal{F}_c$ and equations $E = E_b \cup E_c$. The set of all pure d-terms is $\mathcal{T}(\mathcal{F}_d)$ and the set of all pure a-terms is $\mathcal{T}(\mathcal{F}_a, \mathcal{X} \cup \mathcal{X}_w)$. A term is pure if it is a pure e-term and the set of all pure e-terms will be denoted by $\mathcal{D}_e$. For a term u,

$$\mathsf{root}(u) = \begin{cases} f & \text{if } u = f(u_1, \ldots, u_n) \\ u & \text{if } u \text{ is an atom} \end{cases}$$

and $\mathsf{domain}(u) = e$ iff $\mathsf{root}(u) \in \mathcal{D}_e$. A *term context* over signature $\mathcal{F}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X}_h)$ where $\mathcal{X}_h$ is a special set of variables called holes. Will will use $\_^1, \_^2, \ldots$ to denote holes. A term context H is called non-empty if $H \notin \mathcal{X}_h$. A pure d-term context is a non-empty term context $H \in \mathcal{T}(\mathcal{F}_d, \mathcal{X}_h)$. For a non-empty term context, define $\mathsf{domain}(H) = \mathsf{domain}(\mathsf{root}(H))$. We will write $H[\_^1, \ldots, \_^k]$ to denote a term context with variables $\_^1, \ldots, \_^k$ and $H[u_1, \ldots, u_k]$ to be the term that results from replacing, in H, the variable $\_^j$ by the term u_j for all $j \in \{1, \ldots, j\}$. For a pure d-term context H and a set of terms $u_1, \ldots, u_k$ we will write $H[[u_1, \ldots, u_k]]$ if $\mathsf{domain}(H) \neq \mathsf{domain}(u_j)$ for all $j \in \{1, \ldots, k\}$. The terms $u_1, \ldots, u_k$ are called *alien* subterms of H. Given a set of frame variables $\mathcal{X}_w$, let $\mathcal{X}_w^d = \{w_{i,[\ell]} \mid w_{i,[\ell]} \in \mathcal{X}_w \wedge [\ell] \in \mathcal{L}_d\}$. For the remainder of this section, let $\tilde{s} = s_1, \ldots, s_\ell \in \mathcal{T}(\mathcal{F}_b \cup \mathcal{F}_c)$ be a sequence of terms and $\tilde{n} = n_1, \ldots, n_\ell$ be a sequence of fresh private names such that $\mathsf{domain}(s_i) = \mathsf{domain}(n_i)$ and $n_i = n_j$ iff $s_i =_E s_j$ for all $i, j \in \{1, \ldots, \ell\}$.

Definition 5.1 *Define the function $R^{\tilde{n}}_{\tilde{s},d} : \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{F})$ as follows.*

$$R^{\tilde{n}}_{\tilde{s},d}(u) = \begin{cases} n_i & \text{if } \mathsf{root}(u) \notin \mathcal{F}_d \text{ and } u =_E s_i \text{ for } i \in \{1, \ldots, \ell\} \\ f(R^{\tilde{n}}_{\tilde{s},d'}(u_1), \ldots, R^{\tilde{n}}_{\tilde{s},d'}(u_k)) & \text{otherwise if } u = f(u_1, \ldots, u_k) \text{ and } f \in \mathcal{F}_{d'} \\ u & \text{otherwise} \end{cases}$$

Definition 5.2 *Let $\mathsf{col} : \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{F})$ be a function such that*

$$\mathsf{col}(u) = \begin{cases} u & \text{if } u \text{ is an atom} \\ v_i & \text{if } u = f(u_1, \ldots, u_\ell) \text{ is collapsable to } v_i \\ f(\mathsf{col}(u_1), \ldots, \mathsf{col}(u_\ell)) & \text{otherwise} \end{cases}$$

where a term $f(u_1, \ldots, u_\ell)$ is collapsable to v_i if $f(\mathsf{col}(u_1), \ldots, \mathsf{col}(u_\ell)) = H[[v_1, \ldots, v_k]]$ and $H[[n_1, \ldots, n_k]] =_{E_d} n_i$ where $n_1, \ldots, n_k \in \mathcal{N}^d_{\mathsf{priv}}$ are fresh names such that $n_i = n_j$ iff $v_i =_E v_j$ for all $i, j \in \{1, \ldots, \ell\}$ and $d = \mathsf{domain}(H)$.

We will assume a total order $<_t$ on terms from $\mathcal{T}(\mathcal{F})$. In case 2 of Definition 5.2, if $u = f(u_1, \ldots, u_\ell)$ is collapsible to v_i and v_j for $i, j \in \{1, \ldots, \ell\}$ then $\mathsf{col}(u) = v_i$ if $v_i <_t v_j$. That is, col is a deterministic function. We can easily extend col to a substitution σ by requiring $\mathsf{col}(\sigma)(x) = \mathsf{col}(\sigma(x))$ for all $x \in \mathsf{dom}(\sigma)$.

Lemma 5.2 ([106]) *For any term $u \in \mathcal{T}(\mathcal{F})$, $\mathsf{col}(u) =_E u$.*

Lemma 5.3 ([106]) *If $f \in \mathcal{F}_d$ and $u_1, \ldots, u_\ell \in \mathcal{T}(\mathcal{F})$ then*

$$\mathsf{col}(f(u_1, \ldots, u_\ell)) = \mathsf{col}(f(\mathsf{col}(u_1), \ldots, \mathsf{col}(u_\ell))).$$

Lemma 5.4 ([106]) *(Fundamental Collapse Lemma) Let $u_1, u_2 \in \mathcal{T}(\mathcal{F})$. If $u_1 =_E u_2$, then the following hold.*

1. $\mathsf{col}(u_1) = H_1[[v_1, \ldots, v_k]]$, $\mathsf{col}(u_2) = H_2[[v_{k+1}, \ldots, v_{k+\ell}]]$ *and* $\mathsf{domain}(H_1) = \mathsf{domain}(H_2)$
2. *If $d \in \mathsf{domain}(H_1)$ and $n_1, \ldots, n_{k+\ell}$ are fresh names such that $n_i = n_j$ iff $v_i =_E v_j$ for all $1 \leq i, j \leq k + \ell$ then $H_1[n_1, \ldots, n_k] =_{E_d} H_2[n_{k+1}, \ldots, n_{k+\ell}]$.*

Lemma 5.5 ([106]) *Let $u, v \in \mathcal{T}(\mathcal{F})$. If $u =_E v$, then $R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(u)) =_E R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(v))$.*

Lemma 5.6 ([106]) *Let φ be a frame such that $u \in \mathcal{T}(\mathcal{F})$ for all $u \in \mathsf{ran}(\varphi)$. If $r \in \mathcal{T}(\mathcal{F}, \mathsf{dom}(\varphi))$ is such that $\varphi \not\vdash^{r'} s_i$ for every $i \in \{1, \ldots, \ell\}$ and sub-recipe r' of r then $rR^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(\varphi)) =_E R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(r\varphi))$.*

By our assumption that E is stable by replacement of names by equivalent terms, we have the following.

Definition 5.3 *Let $u_1, u_2 \in \mathcal{T}(\mathcal{F})$ and $n \in \mathsf{st}(u_1) \cup \mathsf{st}(u_2)$. If $u_1 =_E u_2$ and $v_1, v_2 \in \mathcal{T}(\mathcal{F})$ are such that $v_1 =_E v_2$ then $u_1\{n \mapsto v_1\} =_E u_2\{n \mapsto v_2\}$.*

Lemma 5.7 *Let φ and φ' be frames such that $\mathsf{dom}(\varphi) = \mathsf{dom}(\varphi')$ and the following hold.*

1. *$w_{i,[\ell]}\varphi' = R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(w_{i,[\ell]}\varphi))$ for all $w_{i,[\ell]} \in \mathsf{dom}(\varphi)$.*
2. *$u \in \mathcal{T}(\mathcal{F})$ for all $u \in \mathsf{ran}(\varphi) \cup \mathsf{ran}(\varphi')$*

If $\varphi \not\vdash \tilde{s}$ and $\varphi' \not\vdash \tilde{s}$ then $\varphi \equiv \varphi'$.

Proof. We show that for all $r_1, r_2 \in \mathcal{T}(\mathcal{F}, \mathsf{dom}(\varphi))$, $r_1\varphi =_E r_2\varphi$ iff $r_1\varphi' =_E r_2\varphi'$. Assume $r_1\varphi =_E r_2\varphi$. By Lemma 5.5, $R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(r_1\varphi)) =_E R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(r_2\varphi))$. Because $\varphi \not\vdash \tilde{s}$, we can apply Lemma 5.6 to both sides, yielding $r_1R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(\varphi)) =_E r_2R^{\tilde{n}}_{\tilde{s},d}(\mathsf{col}(\varphi))$. By the definition of φ', we have $r_1\varphi' =_E r_2\varphi'$ as desired. The other direction follows by Definition 5.3.

5.3 INDISTINGUISHABILITY IN PRODUCT POMDPS

We first fix some notation. Let $\mathcal{M}_i = (Z_i, z_i^s, \mathsf{Act}_i, \Delta_i, \mathcal{O}_i, \mathsf{obs}_i)$ be a POMDP for $i \in \{1, 2\}$. We will assume that $Z_1 \cap Z_2 = \emptyset$ and $\mathsf{Act}_1 \cap \mathsf{Act}_2 = \emptyset$. The asynchronous product of $\mathcal{M}_1$ and $\mathcal{M}_2$, denoted $\mathcal{M}_1 \otimes \mathcal{M}_2$, is the POMDP $(Z, z^s, \mathsf{Act}, \Delta, \mathcal{O}, \mathsf{obs})$ where $Z = \{(z_1, z_2) \mid z_1 \in$

$Z_1 \wedge z_2 \in Z_2\}$, $z^s = (z_1^s, z_2^s)$, $\mathsf{Act} = \mathsf{Act}_1 \cup \mathsf{Act}_2$, $\mathcal{O} = \{(o_1, o_2) \mid o_1 \in \mathcal{O}_1 \wedge o_2 \in \mathcal{O}_2\}$, $\mathsf{obs}((z_1, z_2)) = (\mathsf{obs}_1(z_1), \mathsf{obs}_2(z_2))$ and Δ is defined below.

$$\Delta((z_1, z_2), \alpha)((z_1', z_2')) = \begin{cases} \Delta_1(z_1, \alpha)(z_1') & \text{if } z_2 = z_2' \wedge \alpha \in \mathsf{Act}_1 \\ \Delta_2(z_2, \alpha)(z_2') & \text{if } z_1 = z_1' \wedge \alpha \in \mathsf{Act}_2 \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{M}$, $\mathcal{M}_1$ and $\mathcal{M}_2$ be as above. We will assume that $\mathsf{Exec}(\mathcal{M})$ is finite and every execution is of finite length. For a state $z = (z_1, z_2) \in Z$ (resp. an observation $o = (o_1, o_2) \in \mathcal{O}$), its projection onto $\mathcal{M}_i$ is $\pi_i(z) = z_i$ (resp. $\pi_i(o) = o_i$). We can lift this projection to executions (resp. traces) by the following inductive definition. For an execution $\rho \in \mathsf{Exec}(\mathcal{M})$ of the form $\rho = \rho_0 \xrightarrow{\alpha} z'$ let $\pi_i(\rho) = \pi_i(\rho) \xrightarrow{\alpha} \pi_i(z')$ if $\alpha \in \mathsf{Act}_i$ and otherwise let $\pi_i(\rho) = \pi_i(\rho)$. Likewise, for a trace $\overline{o} \in (\mathcal{O} \cdot \mathsf{Act}) \cdot \mathcal{O}^*$ of the form $\overline{o} = \overline{o}_0 \alpha o'$ let $\pi_i(\overline{o}) = \pi_i(\overline{o}_0\alpha)\pi_i(o')$ if $\alpha \in \mathsf{Act}_i$ and otherwise let $\pi_i(\overline{o}) = \pi_i(\overline{o}_0)$. Let $\mathcal{A}$ be an adversary and $\mathcal{M}^{\mathcal{A}} = (Z_0, z_0^s, \Delta^{\mathcal{A}})$. For a state $z \in Z_0$ and $\rho \in \mathsf{Exec}(\mathcal{M}^{\mathcal{A}})$, let $\mathsf{prob}_{\mathcal{M}}^{z}(\rho, \mathcal{A})$ be the measure of the event ρ starting from state z in $\mathcal{M}^{\mathcal{A}}$. For an observation o and trace $\overline{o}$, let $\mathsf{prob}_{\mathcal{M}}^{o}(\overline{o}, \mathcal{A}) = \sum_{j=1}^{m} \mathsf{prob}_{\mathcal{M}}^{z_j}(\rho_j, \mathcal{A})$ where $\rho_1, \ldots, \rho_m \in \mathsf{Exec}(\mathcal{M}^{\mathcal{A}})$ are executions such that $\mathsf{tr}(\rho_j) = \overline{o}$ and the initial state z_j of ρ_j is such that $\mathsf{obs}(z_j) = o$. Let $\mathcal{A}$ be an adversary and $\overline{o}$ be a trace. Define $\mathsf{proj}_i^{\overline{o}}(\mathcal{A})$ as follows. For any trace $\overline{o}_0$, $\mathsf{proj}_i^{\overline{o}}(\mathcal{A})(\overline{o}_0) = \mathcal{A}(\overline{o}_1)$ if $\overline{o}_1$ is a maximal trace such that $\overline{o}_1 \preceq \overline{o}$ and $\pi_i(\overline{o}_1) = \overline{o}_0$.

Lemma 5.8 *Let $\mathcal{M}$, $\mathcal{M}_1$ and $\mathcal{M}_2$ be as above. Let $\overline{o}$ be a trace, $\mathcal{A}$ be an adversary and $\mathcal{A}_i = \mathsf{proj}_i^{\overline{o}}(\mathcal{A})$. We have $\mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{\mathcal{M}_1}(\pi_1(\overline{o}), \mathcal{A}_1) \times \mathsf{prob}_{\mathcal{M}_2}(\pi_2(\overline{o}), \mathcal{A}_2)$.*

Proof. The proof is by induction on the length of $\overline{o}$. For the base case, when $\overline{o} = o$, we have $\mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(o, \mathcal{A}) = 1$ and $\mathsf{prob}_{\mathcal{M}_i}(\pi_i(\overline{o}), \mathcal{A}_i) = \mathsf{prob}_{\mathcal{M}_i}(\pi_i(o), \mathcal{A}_i) = 1$. For the induction step, let $\overline{o} = o\alpha\overline{o}_0$. We can assume without loss of generality that $\alpha \in \mathsf{Act}_1$. Let $o = (o_1^s, o_2^s)$. Then the first observation of $\overline{o}_0$ is $o_0 = (o', o_2^s)$. Let $\rho_1, \ldots, \rho_m \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ be the executions such that $\mathsf{tr}(\rho_j) = \overline{o}$ for $j \in \{1, \ldots, m\}$. Further let $\rho_j = z^s \xrightarrow{\alpha} \rho_j'$ and observe that every initial state z_j' of ρ_j' is such that $\mathsf{obs}(z_j') = o_0$. We have the following.

$$\begin{aligned} \mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(\overline{o}, \mathcal{A}) &= \textstyle\sum_{j=1}^{m} \mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(\rho_j, \mathcal{A}) \\ &= \textstyle\sum_{j=1}^{m} \Delta^{\mathcal{A}}(z^s)(z_j') \times \mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(\rho_j', \mathcal{A}) \\ &= \textstyle\sum_{j=1}^{m} \Delta^{\mathcal{A}}(z^s)(z_j') \times \sum_{j=1}^{m} \mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(\rho_j', \mathcal{A}) \\ &= \textstyle\sum_{j=1}^{m} \Delta^{\mathcal{A}}(z^s)(z_j') \times \mathsf{prob}^{o_0}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(\overline{o}_0, \mathcal{A}) \end{aligned}$$

We will assume without loss of generality that $z_1', \ldots, z_m'$ are distinct states. Let $\mathcal{M}_j'$ be the same as $\mathcal{M}_1$ with the exception that its initial state is z_j'. Further, let $\mathcal{A}'$ be the adversary such that $\mathcal{A}'(\overline{o}') = \mathcal{A}(o\alpha\overline{o}')$. If $\mathcal{A}_i' = \mathsf{proj}_i^{\overline{o}}(\mathcal{A}')$, then we have the following.

$$\begin{aligned}
\mathsf{prob}^{o_0}_{\mathcal{M}_1\otimes\mathcal{M}_2}(\overline{o}_0,\mathcal{A}) &= \textstyle\sum_{j=1}^{m}\mathsf{prob}_{\mathcal{M}'_j\otimes\mathcal{M}_2}(\overline{o}_0,\mathcal{A}') \\
&= \textstyle\sum_{j=1}^{m}\mathsf{prob}_{\mathcal{M}'_j}(\pi_i(\overline{o}_0),\mathcal{A}'_1)\times\mathsf{prob}_{\mathcal{M}_2}(\pi_2(\overline{o}_0),\mathcal{A}'_2) \\
&= \mathsf{prob}^{o'}_{\mathcal{M}_1}(\pi_1(\overline{o}_0),\mathcal{A}_1)\times\mathsf{prob}^{o^s_2}_{\mathcal{M}_2}(\pi_2(\overline{o}_0),\mathcal{A}_2)
\end{aligned}$$

Let $\rho''_1,\ldots,\rho''_m$ be the executions of $\mathcal{M}_1^{\mathcal{A}_1}$ such that $\pi_1(\rho''_j)=\rho'_j$ and $\pi_1(z'_j)=z''_j$. Then we have $\mathsf{prob}^{o'}_{\mathcal{M}_1}(\pi_1(\overline{o}_0),\mathcal{A}_1)=\sum_{j=1}^{m}\mathsf{prob}^{z''_j}_{\mathcal{M}_1}(\rho''_j,\mathcal{A}_1)$. Furthermore, because $\alpha\in\mathsf{Act}_1$, it must be the case that $\Delta^{\mathcal{A}}(z^s)(z''_m)=\Delta_1^{\mathcal{A}_1}(\pi_1(z^s))(\pi_1(z''_m))$ and $\mathsf{prob}^{o^s_2}_{\mathcal{M}_2}(\pi(\overline{o}_0),\mathcal{A}_2)=\mathsf{prob}_{\mathcal{M}_2}(\pi(\overline{o}_0),\mathcal{A}_2)$. Putting everything together, we have $\mathsf{prob}_{\mathcal{M}_1\otimes\mathcal{M}_2}(\overline{o},\mathcal{A})$

$$\begin{aligned}
&= \textstyle\sum_{j=1}^{m}\Delta^{\mathcal{A}}(z^s)(z'_j)\times\mathsf{prob}^{o_0}_{\mathcal{M}_1\otimes\mathcal{M}_2}(\overline{o}_0,\mathcal{A}) \\
&= \textstyle\sum_{j=1}^{m}\Delta^{\mathcal{A}}(z^s)(z'_j)\times\mathsf{prob}^{o'}_{\mathcal{M}_1}(\pi_1(\overline{o}_0),\mathcal{A}_1)\times\mathsf{prob}^{o^s_2}_{\mathcal{M}_2}(\pi_2(\overline{o}_0),\mathcal{A}_2) \\
&= \textstyle\sum_{j=1}^{m}\Delta_1^{\mathcal{A}_1}(\pi_1(z^s))(\pi_1(z''_j))\times\sum_{j=1}^{m}\mathsf{prob}^{z''_j}_{\mathcal{M}_1}(\rho''_j,\mathcal{A}_1)\times\mathsf{prob}^{o^s_2}_{\mathcal{M}_2}(\pi_2(\overline{o}_0),\mathcal{A}_2) \\
&= \textstyle\sum_{j=1}^{m}\Delta_1^{\mathcal{A}_1}(\pi_1(z^s))(\pi_1(z''_j))\times\mathsf{prob}^{z''_j}_{\mathcal{M}_1}(\rho''_j,\mathcal{A}_1)\times\mathsf{prob}_{\mathcal{M}_2}(\pi_2(\overline{o}_0),\mathcal{A}_2) \\
&= \mathsf{prob}_{\mathcal{M}_1}(\pi_1(\overline{o}_0),\mathcal{A}_1)\times\mathsf{prob}_{\mathcal{M}_2}(\pi_2(\overline{o}_0),\mathcal{A}_2).
\end{aligned}$$

Lemma 5.9 *Let $\mathcal{M},\mathcal{M}_1$ and $\mathcal{M}'_1$ be POMDPs such that the states and actions of $\mathcal{M}$ and $\mathcal{M}_1$ (resp. $\mathcal{M}'_1$) are disjoint. If $\mathcal{M}_1\approx\mathcal{M}'_1$ then $\mathcal{M}_1\otimes\mathcal{M}\approx\mathcal{M}'_1\otimes\mathcal{M}$.*

Proof. Assume for a contradiction that $\mathcal{M}_1\otimes\mathcal{M}\not\approx\mathcal{M}'_1\otimes\mathcal{M}$. By definition, there exists a trace $\overline{o}$ and adversary $\mathcal{A}$ such that $\mathsf{prob}_{\mathcal{M}_1\otimes\mathcal{M}}(\overline{o},\mathcal{A})\neq\mathsf{prob}_{\mathcal{M}'_1\otimes\mathcal{M}}(\overline{o},\mathcal{A})$. Let $\mathcal{A}_i=\mathsf{proj}_i^{\overline{o}}(\mathcal{A})$ for $i\in\{1,2\}$. By Lemma 5.8,

$$\mathsf{prob}_{\mathcal{M}_1}(\pi_1(\overline{o}),\mathcal{A}_1)\times\mathsf{prob}_{\mathcal{M}}(\pi_2(\overline{o}),\mathcal{A}_2)\neq\mathsf{prob}_{\mathcal{M}'_1}(\pi_1(\overline{o}),\mathcal{A}_1)\times\mathsf{prob}_{\mathcal{M}}(\pi_2(\overline{o}),\mathcal{A}_2).$$

Because $\mathsf{prob}_{\mathcal{M}_1\otimes\mathcal{M}}(\overline{o},\mathcal{A})\neq\mathsf{prob}_{\mathcal{M}'_1\otimes\mathcal{M}}(\overline{o},\mathcal{A})$ we have $\mathsf{prob}_{\mathcal{M}}(\pi_2(\overline{o}),\mathcal{A}_2)\neq 0$. That is,

$$\mathsf{prob}_{\mathcal{M}_1}(\pi_1(\overline{o}),\mathcal{A}_1)\neq\mathsf{prob}_{\mathcal{M}'_1}(\pi_1(\overline{o}),\mathcal{A}_1)$$

and hence $M_1\not\approx M'_1$, contradiction.

The following is a consequence of Lemma 5.9.

Lemma 5.10 *Let $\mathcal{M}_1$, $\mathcal{M}'_1$, $\mathcal{M}_2$ and $\mathcal{M}'_2$ be POMDPs such that the states and actions of $\mathcal{M}_1$ (resp. $\mathcal{M}'_1$) and $\mathcal{M}_2$ (resp. $\mathcal{M}'_2$) are disjoint. If $\mathcal{M}_1\approx\mathcal{M}'_1$ and $\mathcal{M}_2\approx\mathcal{M}'_2$, then $\mathcal{M}_1\otimes\mathcal{M}_2\approx\mathcal{M}'_1\otimes\mathcal{M}'_2$.*

5.4 SINGLE-SESSION PROTOCOLS OVER DISJOINT DATA

This section is dedicated to the proof of Lemma 5.1. Throughout the section, we will assume an equational theory $(\mathcal{F}, E)$ with private names $\mathcal{N}_{\mathsf{priv}}$ and public names $\mathcal{N}_{\mathsf{pub}}$. We will also assume two disjoint set of labels $\mathcal{L}$ that is partitioned into two disjoint sets $\mathcal{L}_b$ and $\mathcal{L}_c$. As before, $\mathcal{L}$ is equipped with an equivalence relation $\sim$.

Lemma 5.11 *Let $\varphi_0, \varphi_0', \varphi_1$ and φ_1' be frames and $\mathcal{N}_0, \mathcal{N}_1 \subseteq \mathcal{N}_{\mathsf{priv}}$ be sets of private names such that $\mathcal{N}_0 \cap \mathcal{N}_1 = \emptyset$, $\mathcal{N}_0 \cup \mathcal{N}_1 = \mathcal{N}_{\mathsf{priv}}$ and the following hold.*

1. *$\forall u \in \mathsf{ran}(\varphi_0) \cup \mathsf{ran}(\varphi_0')$ and $\forall w_{i,[\ell]} \in \mathsf{dom}(\varphi_0) \cup \mathsf{dom}(\varphi_0')$, $u \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_1)$ and $\ell \in \mathcal{L}_b$.*
2. *$\forall u \in \mathsf{ran}(\varphi_1) \cup \mathsf{ran}(\varphi_1')$ and $\forall w_{i,[\ell]} \in \mathsf{dom}(\varphi_1) \cup \mathsf{dom}(\varphi_1')$, $u \in \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_0)$ and $\ell \in \mathcal{L}_c$*
3. *$\varphi = \varphi_0 \cup \varphi_1$ and $\varphi' = \varphi_0' \cup \varphi_1'$*

Then $\varphi \equiv \varphi'$ iff $\varphi_0 \equiv \varphi_0'$ and $\varphi_1 \equiv \varphi_1'$.

Proof. "$\Rightarrow$" We show the contrapositive. Assume without loss of generality that $\varphi_0 \not\equiv \varphi_0'$. That is, $\mathsf{dom}(\varphi) \neq \mathsf{dom}(\varphi')$ or there exist recipes r_1, r_2 such that $r_1\varphi_0 =_E r_2\varphi_0$ and $r_1\varphi_0' \neq_E r_2\varphi_0'$ (or vice versa, in which case the result follows by a similar argument). If $\mathsf{dom}(\varphi) \neq \mathsf{dom}(\varphi')$ then the result is immediate. Otherwise the free variables in r_1, r_2 must come from $\mathsf{dom}(\varphi_0) = \mathsf{dom}(\varphi_0')$ and thus they have the form $w_{i,[\ell]}$ for $\ell \in \mathcal{L}_b$. By definition, for all such $w_{i,[\ell]}$, we have $w_{i,[\ell]}\varphi_0 = w_{i,[\ell]}\varphi$ and $w_{i,[\ell]}\varphi_0' = w_{i,[\ell]}\varphi'$. That is, $r_1\varphi =_E r_2\varphi$ and $r_1\varphi' \neq_E r_2\varphi'$, which means that $\varphi \not\equiv \varphi'$.

"$\Leftarrow$" We show the contrapositive. Assume that $\varphi \not\equiv \varphi'$. If $\mathsf{dom}(\varphi) \neq \mathsf{dom}(\varphi')$ then the result is immediate. Otherwise there exist recipes r_b, r_c such that $r_b\varphi =_E r_c\varphi$ and $r_b\varphi' \neq_E r_c\varphi'$ (or vice versa, in which case the result follows by a similar argument). Again, if $r_b, r_c \in \mathsf{dom}(\varphi)$ then the result is immediate, so we will assume that for all $w_{i,[\ell_1]}, w_{j,[\ell_2]} \in \mathsf{dom}(\varphi)$, we have $w_{i,[\ell_1]}\varphi =_E w_{j,[\ell_2]}\varphi$ iff $w_{i,[\ell_1]}\varphi' =_E w_{j,[\ell_2]}\varphi'$. Let $\xi : \mathcal{T}(\mathcal{F}) \to \mathcal{T}(\mathcal{F})$ be a function such that $\xi(u_1) = \xi(u_2)$ iff $u_1 =_E u_2$ and $\xi(u_1) \in [u_1]$. Let $\tilde{\varphi}$ (resp. $\tilde{\varphi}'$) be the frame that results from replacing every $u \in \mathsf{ran}(\varphi)$ (resp. $u \in \mathsf{ran}(\varphi')$) by $\xi(u)$. Clearly, for any recipe r, $r\varphi =_E r\tilde{\varphi}$ and $r\varphi' =_E r\tilde{\varphi}'$. Let $\mathsf{rename}_0 : \mathcal{N}_0 \to M_0$ and $\mathsf{rename}_1 : \mathcal{N}_1 \to M_1$ be bijections such that $M_0 \cap M_1 = \emptyset$, $M_0, M_1 \subset \mathcal{N}_{\mathsf{pub}}$ and no names in M_0, M_1 occur in φ, φ'. Given a term u, we will write $\mathsf{rename}_0(u)$ (resp. $\mathsf{rename}_1(u)$) to denote the term that results from replacing in u, every $k \in \mathcal{N}_0$ (resp. $k \in \mathcal{N}_1$) by $\mathsf{rename}_0(k)$ (resp. $\mathsf{rename}_1(k)$).

For a recipe r, let r^0 (resp. r^1) be the recipe that that results from replacing every sub-recipe of the form $u = w_{i,[\ell]}$ for $\ell \in \mathcal{L}_c$ (resp. $\ell \in \mathcal{L}_b$) in r by the sub-recipe $\mathsf{rename}_1(u\varphi)$ (resp.

$\mathsf{rename}_0(u\varphi)$). By Definition 5.3, we have $r\tilde{\varphi} =_E r^0\tilde{\varphi} =_E r^1\tilde{\varphi}$ and $r\tilde{\varphi}' =_E r^0\tilde{\varphi}' =_E r^1\tilde{\varphi}'$. Given this, it must be the case that either

$$r_b^0\tilde{\varphi} =_E r_c^0\tilde{\varphi} \wedge r_b^0\tilde{\varphi}' \neq_E r_c^0\tilde{\varphi}' \text{ or } r_b^1\tilde{\varphi} =_E r_c^1\tilde{\varphi} \wedge r_b^1\tilde{\varphi}' \neq_E r_c^1\tilde{\varphi}'.$$

Because the free variables of r_b^0, r_c^0 are over $\mathsf{dom}(\varphi_0)$ and the free variables of r_b^1, r_c^1 are over $\mathsf{dom}(\varphi_1)$ we can conclude that either $\varphi_0 \not\equiv \varphi_0'$ or $\varphi_1 \not\equiv \varphi_1'$.

For a set of frame variables $\mathcal{X}_w$ and a set of labels $L \subseteq \mathcal{L}$ closed under $\sim$, we write $\mathcal{X}_w^L$ to denote the set $\{w_{i,[\ell]} \mid w_{i,[\ell]} \subseteq \mathcal{X}_w \text{ and } \ell \in L\}$.

Definition 5.4 *Let L be a set of labels closed under $\sim$. An adversary $\mathcal{A}$ for a process Q is said to be pure with respect to L if whenever $\mathcal{A}$ chooses an action $(r, [\ell])$, we have $r \in (\mathcal{F}, \mathcal{X}_w^L)$.*

Lemma 5.12 *Let P, P' and Q be processes such that $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \mathsf{vars}(P') \cap \mathsf{vars}(Q) = \emptyset$. If $P \approx P'$ then $P \mid Q \approx P' \mid Q$.*

Proof. We show the contrapositive, that is $P \mid Q \not\approx P' \mid Q$ implies $P \not\approx P'$. The proof is by induction on the number of probabilistic choices in Q. For the base case, let Q be a process that doesn't contain probabilistic choice. Define $S = \{k \mid \nu k \text{ occurs in } Q\}$ to be the set of all variable names that can be bound by the ν operator in Q. Further let $M \subset \mathcal{N}_{\mathsf{pub}}$ be a set of names not occurring in P, P' and Q and $\mathsf{rename} : S \to M$ be a bijection. Define Q' as the process that results from replacing, in Q, every occurrence of an atomic action νk by $(k := \mathsf{rename}(k))$. If $P \mid Q \not\approx P' \mid Q$ then $P \mid Q' \not\approx P' \mid Q'$, which is a straightforward consequence of Definition 5.3.

We can assume, by convention, that P, P' are labeled by $\mathcal{L}_b$ and Q is labeled by $\mathcal{L}_c$. Let $\mathcal{A}$ be an adversary and $\overline{o}$ be a trace such that $\mathsf{prob}_{P|Q'}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{P'|Q'}(\overline{o}, \mathcal{A})$. From $\mathcal{A}$, we define an adversary $\mathcal{A}'$ for $P \mid Q'$ and $P' \mid Q'$ that is pure with respect to $\mathcal{L}_b$ as follows. Let $\rho \in \mathsf{Exec}([\![P \mid Q']\!]^{\mathcal{A}} \cup [\![P' \mid Q']\!]^{\mathcal{A}})$ be such that $\mathsf{tr}(\rho) = \overline{o}$ and $\mathsf{last}(\rho) = (R, \varphi, \sigma)$. We will define mappings $\Theta_\rho : \mathsf{dom}(\sigma) \cap \mathsf{vars}(Q') \to \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathsf{dom}(\varphi))$ and $\Phi_\rho : \mathsf{dom}(\varphi) \to \mathcal{T}(\mathcal{F} \setminus \mathcal{N}_{\mathsf{priv}}, \mathsf{dom}(\varphi))$ by induction on the length of ρ. If ρ contains no actions then $\mathsf{dom}(\sigma) = \mathsf{dom}(\varphi) = \emptyset$. Inductively let $\rho = \rho_0 \xrightarrow{\alpha} z$. We distinguish 4 cases.

case 1: α does not execute an output action or an input/assignment action that binds a variable in $\mathsf{dom}(\sigma) \cap \mathsf{vars}(Q')$. Define $\Theta_\rho = \Theta_{\rho_0}$ and $\Phi_\rho = \Phi_{\rho_0}$.

case 2: α executes output action of the form $\mathtt{out}(u)$ where $u \in \mathcal{T}(\mathcal{F}, \mathsf{dom}(\sigma))$. Define $\Theta_\rho = \Theta_{\rho_0}$. The output action must bind a frame variable $w_{j,[\ell]}$ to the value $u\sigma$. If $\ell \in \mathcal{L}_c$ then $\Phi_\rho(w_{j,[\ell]}) = u\Theta_{\rho_0}$. Otherwise, if $\ell \in \mathcal{L}_b$ then $\Phi_\rho(w_{j,[\ell]}) = w_{j,[\ell]}$.

case 3: α executes a action of the form $\mathtt{in}(x)$ from Q'. The action α must be of the form $(r, [\ell])$. Define $\Theta_\rho(x) = r\Phi_{\rho_0}$ and $\Phi_\rho = \Phi_{\rho_0}$.

case 4: α executes a action of the form $(x := u)$ from Q' where $u \in \mathcal{T}(\mathcal{F}, \mathsf{dom}(\sigma))$. Define $\Theta_\rho(x) = u\Theta_{\rho_0}$ and $\Phi_\rho = \Phi_{\rho_0}$.

We extend Φ_ρ to actions by requiring that $\Phi_\rho(\tau, [\ell]) = (\tau, [\ell])$ and $\Phi_\rho(r, [\ell]) = (r\Phi_\rho, [\ell])$. Observe that because Q doesn't contain probabilistic choice, for any two executions $\rho, \rho' \in \mathsf{Exec}([\![P \mid Q']\!]^{\mathcal{A}}) \cup \mathsf{Exec}([\![P' \mid Q']\!]^{\mathcal{A}})$ such that $\mathsf{tr}(\rho) = \mathsf{tr}(\rho')$, we have $\Phi_\rho(\alpha) = \Phi_{\rho'}(\alpha)$. Thus, for a trace $\overline{o}$ and action α, we can define $\Phi_{\overline{o}}$ as $\Phi_\rho(\alpha)$ where ρ is any execution such that $\mathsf{tr}(\rho) = \overline{o}$. Now if $\overline{o}_0 = \mathsf{obs}(z_0)\alpha_1 \dots \alpha_{k-1}\mathsf{obs}(z_{k-1})$ is a prefix of $\overline{o}$ and $\mathcal{A}(\overline{o}_0) = \alpha_k$ let $\mathcal{A}'(\mathsf{obs}(z_0)\Phi'_{\overline{o}}(\alpha_0) \dots \Phi_{\overline{o}}(\alpha_{k-1})\mathsf{obs}(z_k)) = \Phi_{\overline{o}}(\alpha_k)$.

By a simple induction on the length of ρ, one can show that $\rho \in \mathsf{Exec}([\![P \mid Q']\!]^{\mathcal{A}})$ iff $\rho \in \mathsf{Exec}([\![P \mid Q']\!]^{\mathcal{A}'})$ and $\mathsf{prob}_{P|Q}(\rho, \mathcal{A}) = \mathsf{prob}_{P|Q'}(\rho, \mathcal{A}')$. Likewise, we have $\rho \in \mathsf{Exec}([\![P' \mid Q']\!]^{\mathcal{A}})$ iff $\rho \in \mathsf{Exec}([\![P' \mid Q']\!]^{\mathcal{A}'})$ and $\mathsf{prob}_{P'|Q'}(\rho, \mathcal{A}) = \mathsf{prob}_{P'|Q'}(\rho, \mathcal{A}')$. The preceding facts imply that $\mathcal{A}'$ is an adversary for $P \mid Q'$ and $P' \mid Q'$ such that $\mathsf{prob}_{P|Q'}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{P|Q'}(\overline{o}, \mathcal{A}')$ and $\mathsf{prob}_{P'|Q'}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{P'|Q'}(\overline{o}, \mathcal{A}')$. That is $\mathsf{prob}_{P|Q'}(\overline{o}, \mathcal{A}') \neq \mathsf{prob}_{P'|Q'}(\overline{o}, \mathcal{A}')$. By our construction, $\mathcal{A}'$ doesn't use an recipes output by Q'. By Lemma 5.11 and Lemma 5.9, we can conclude that $P \not\approx P'$.

For the induction step, let Q' be the result of replacing, in Q, some occurrence of a subprocess $Q_0 +_p Q_1$ by $\mathtt{out}(0)^{\ell_0} \cdot Q_0 +_p \mathtt{out}(1)^{\ell_1} \cdot Q_1$ where $0, 1$ are fresh unary constant symbols. Define a projection π from executions of $P \mid Q'$ (resp. $P' \mid Q'$) to executions of $P \mid Q$ (resp. $P' \mid Q$) inductively as follows. If $\rho \in \mathsf{Exec}([\![P \mid Q']\!])$ (resp. $\rho \in \mathsf{Exec}([\![P' \mid Q']\!])$) contains no actions, then $\pi(P \mid Q', \emptyset, \emptyset) = (P \mid Q, \emptyset, \emptyset)$ (resp. $\pi(P' \mid Q', \emptyset, \emptyset) = (P' \mid Q, \emptyset, \emptyset)$). Inductively, let $\rho = \rho_0 \xrightarrow{\alpha} z$ for $\alpha = (\S, [\ell])$. If $\ell \in \{\ell_0, \ell_1\}$ then $\pi(\rho) = \pi(\rho_0)$. Otherwise $\pi(\rho) = \pi(\rho_0) \xrightarrow{\alpha} z$. The projection π can be extended to traces by requiring that $\pi(\overline{o}) = \mathsf{tr}(\pi(\rho))$. From π we can define an adversary $\mathcal{A}'$ for $P \mid Q'$ (resp. $P' \mid Q'$) from an adversary $\mathcal{A}$ for $P \mid Q$ (resp. $P' \mid Q$) in the following way. For a trace $\overline{o}$, if $\mathcal{A}(\pi(\overline{o})) = (\S, [\ell])$ where ℓ occurs in Q_0 (resp. Q_1) and $\overline{o}$ doesn't contain the action $(\tau, [\ell_0])$ (resp. $(\tau, [\ell_1])$), then $\mathcal{A}'(\overline{o}) = (\tau, [\ell_0])$ (resp. $\mathcal{A}'(\overline{o}) = (\tau, [\ell_1])$). Otherwise $\mathcal{A}'(\overline{o}) = \mathcal{A}(\pi(\overline{o}))$. Clearly we have $\mathsf{prob}_{P|Q}(\rho, \mathcal{A}) = \mathsf{prob}_{P|Q'}(\pi(\rho), \mathcal{A}')$ and $\mathsf{prob}_{P'|Q}(\rho, \mathcal{A}) = \mathsf{prob}_{P'|Q'}(\pi(\rho), \mathcal{A}')$. Furthermore, for any $\rho_1, \rho_2 \in \mathsf{Exec}([\![P \mid Q']\!]^{\mathcal{A}'})$ (resp. $\rho_1, \rho_2 \in \mathsf{Exec}([\![P' \mid Q']\!]^{\mathcal{A}'})$) if $\mathsf{tr}(\rho_1) \neq \mathsf{tr}(\rho_2)$ then $\mathsf{tr}(\pi(\rho_1)) \neq \mathsf{tr}(\pi(\rho_2))$. Because $P \mid Q \not\approx P' \mid Q$ there is an adversary $\mathcal{A}$ and trace $\overline{o}$ such that $\mathsf{prob}_{P|Q}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{P'|Q}(\overline{o}, \mathcal{A})$. Let $\overline{o}_1, \dots, \overline{o}_v$ be the traces of $(P \mid Q')^{\mathcal{A}'}, (P' \mid Q')^{\mathcal{A}'}$ such that $\mathsf{tr}(\overline{o}_j) = \overline{o}$ for all $j \in \{1, \dots, v\}$. By the preceding observations, there is some j such that $\mathsf{prob}_{P|Q'}(\overline{o}_j, \mathcal{A}') \neq \mathsf{prob}_{P'|Q'}(\overline{o}_j, \mathcal{A}')$ and hence $P \mid Q' \not\approx P' \mid Q'$. Let $\rho_0, \rho_1 \in \mathsf{Exec}([\![P \mid Q']\!]^{\mathcal{A}'})$ (resp. $\rho_0, \rho_1 \in \mathsf{Exec}([\![P' \mid Q']\!]^{\mathcal{A}'})$) be executions such that ρ_0 contains the action $(\tau, [\ell_0])$ and ρ_1 contains the action $(\tau, [\ell_1])$. We have $\mathsf{tr}(\rho_0) \neq \mathsf{tr}(\rho_1)$. From this, we can conclude that

$P \mid Q'_0 \not\approx P' \mid Q'_0$ or $P \mid Q'_1 \not\approx P' \mid Q'_1$ where Q'_0 (resp. Q'_1) is the process that results from replacing, in Q, some occurrence of the subprocess $Q_0 +_p Q_1$ by Q_0 (resp. Q_1). By the induction hypothesis, we can conclude that $P \not\approx P'$.

By Lemma 5.12, we have $P \mid Q \approx P' \mid Q$ and $P' \mid Q \approx P' \mid Q'$. The result from Lemma 5.1 follows.

5.5 SINGLE-SESSION PROTOCOLS OVER DISJOINT PRIMITIVES

In this section, we complete the proof of Theorem 5.1. For any two executions ρ_0, ρ_1 we say that ρ_0 is a prefix of ρ_1, denoted $\rho_0 \preceq \rho_1$, if $\rho_1 = \rho_0 \xrightarrow{\alpha_0} \ldots \xrightarrow{\alpha_k} z_k$. The prefix operation $\preceq$ is extended to traces in the natural way. For a process P and an adversary $\mathcal{A}$, we will write $\mathsf{MExec}([\![P]\!]^{\mathcal{A}})$ to denote the subset of $\mathsf{Exec}([\![P]\!]^{\mathcal{A}})$ that contains only maximal executions. A trace $\overline{o}$ is said to be maximal for $[\![P]\!]^{\mathcal{A}}$ if $\overline{o} = \mathsf{tr}(\rho)$ where $\rho \in \mathsf{MExec}([\![P]\!]^{\mathcal{A}})$. The set of all attackers will be denoted by A.

Proposition 5.1 *Let P and Q be processes such that $P \not\approx Q$. There exist an adversary $\mathcal{A}$ and a trace $\overline{o}$ that is maximal for $[\![P]\!]^{\mathcal{A}}$ and $[\![Q]\!]^{\mathcal{A}}$ such that $\mathsf{prob}_P(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_Q(\overline{o}, \mathcal{A})$.*

Definition 5.5 *Let (P_1, P_2) (resp. (Q_1, Q_2)) be a pair of processes and $\mathcal{A}$ be an adversary. Further Let $\Lambda : \mathsf{A} \to \mathsf{A}$ be a function and $\delta : \mathsf{MExec}([\![P_1]\!]^{\mathcal{A}}) \uplus \mathsf{MExec}([\![P_2]\!]^{\mathcal{A}}) \to \mathsf{MExec}([\![Q_1]\!]^{\Lambda(\mathcal{A})}) \uplus \mathsf{MExec}([\![Q_2]\!]^{\Lambda(\mathcal{A})})$ be a bijection such that the following hold.*

1. *For any $\rho \in \mathsf{MExec}([\![P_i]\!]^{\mathcal{A}})$, $\mathsf{prob}_{P_i}(\rho, \mathcal{A}) = \mathsf{prob}_{Q_i}(\delta(\rho), \Lambda(\mathcal{A}))$*
2. *For any $\rho_0, \rho_1 \in \mathsf{MExec}([\![P_1]\!]^{\mathcal{A}}) \uplus \mathsf{MExec}([\![P_2]\!]^{\mathcal{A}})$, $\mathsf{tr}(\rho_0) = \mathsf{tr}(\rho_1)$ iff $\mathsf{tr}(\delta(\rho_0)) = \mathsf{tr}(\delta(\rho_1))$.*

Then (P_1, P_2) is said to be transposable *to (Q_1, Q_2) by Λ.*

Lemma 5.13 *Let (P_1, P_2) be transposable to (Q_1, Q_2). If $P_1 \not\approx P_2$ then $Q_1 \not\approx Q_2$.*

Proof. Assume $P_1 \not\approx P_2$. By Proposition 5.1, there exists and adversary $\mathcal{A}$ and a maximal trace $\overline{o}$ such that $\mathsf{prob}_{P_1}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{P_2}(\overline{o}, \mathcal{A})$. Let $\rho_1, \ldots, \rho_k \in \mathsf{MExec}([\![P_1]\!]^{\mathcal{A}})$ (resp. $\rho'_1, \ldots, \rho'_\ell \in \mathsf{MExec}([\![P_2]\!]^{\mathcal{A}})$) be the executions such that $\mathsf{tr}(\rho_i) = \overline{o}$ (resp. $\mathsf{tr}(\rho'_j) = \overline{o}$) for all $i \in \{1, \ldots, k\}$ (resp. $j \in \{1, \ldots, \ell\}$). By property 2 of Definition 5.5 there exists a trace $\overline{o}_0$ such that for the executions $\delta(\rho_1), \ldots, \delta(\rho_k) \in \mathsf{MExec}([\![Q_1]\!]^{\Lambda(\mathcal{A})})$ (resp. $\delta(\rho'_1), \ldots, \delta(\rho'_\ell) \in \mathsf{MExec}([\![Q_2]\!]^{\Lambda(\mathcal{A})})$) $\mathsf{tr}(\delta(\rho_i)) = \overline{o}_0$ (resp. $\mathsf{tr}(\delta(\rho'_j)) = \overline{o}_0$) for all i (resp. j). By property 1 of Definition 5.5, $\mathsf{prob}_{Q_1}(\overline{o}_0, \Lambda(\mathcal{A})) = \mathsf{prob}_{P_1}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{P_2}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{Q_2}(\overline{o}_0, Q_2^{\Lambda(\mathcal{A})})$.

Figure 5.2 *Interleavings for parallel composition of basic processes.*

$$\frac{D \in \mathcal{I}(D_1 \mid \ldots \mid D_m)}{a \cdot D \in \mathcal{I}(D_1 \mid \ldots \mid D_{i-1} \mid a \mid D_{i+1} \mid \ldots \mid D_m)} \qquad \frac{D \in \mathcal{I}(D_1 \mid \ldots \mid D_i \mid \ldots \mid D_m)}{a \cdot D \in \mathcal{I}(D_1 \mid \ldots \mid a \cdot D_i \mid \ldots \mid D_m)}$$

$$\frac{D' \in \mathcal{I}(D_1 \mid \ldots \mid D'_i \mid \ldots \mid D_m) \quad D'' \in \mathcal{I}(D_1 \mid \ldots \mid D''_i \mid \ldots \mid D_m)}{D' +_p D'' \in \mathcal{I}(D_1 \mid \ldots \mid D'_i +_p D''_i \mid \ldots \mid D_m)}$$

Recall the definition of an atomic process from Section 2.7. We will use a to denote atomic processes. A process is called linear if it can be derived from the grammar $L := a \mid (L \cdot a)$. Let P be a process and $\mathcal{A}$ be an adversary for P. For any $\rho \in \mathsf{Exec}([\![P]\!]^{\mathcal{A}})$, define the linear process $L(\rho)$ inductively as follows. For the base case $L((P, \emptyset, \emptyset)) = \epsilon$. For the inductive case, let $\rho = \rho_0 \xrightarrow{(\S,[\ell])} z_n$. If ℓ is the label of an atomic process a, then $L(\rho) = L(\rho_0) \cdot a$. Otherwise ℓ is the label of a probabilistic choice and $L(\rho) = L(\rho_0) \cdot [T]^{\ell}$.

Proposition 5.2 *Let P be a process, $\mathcal{A}$ be an adversary for P and $\rho \in \mathsf{Exec}([\![P]\!]^{\mathcal{A}})$. There exists an adversary $\mathcal{A}'$ for $L(\rho)$, an execution $\rho' \in \mathsf{Exec}([\![L(\rho)]\!]^{\mathcal{A}'})$, a frame φ and a substitution σ such that $\mathsf{last}(\rho) = (Q, \varphi, \sigma)$ and $\mathsf{last}(\rho') = (Q', \varphi, \sigma)$.*

A process is called deterministic if it can be derived from the following grammar.

$$D \ ::= \ a \,\Big|\, D +_p D \,\Big|\, a \cdot D$$

The concatenation of two deterministic process, denoted $D_1 \circ D_2$, is defined below.

$$D_1 \circ D_2 = \begin{cases} a \cdot D_2 & \text{if } D_1 = a \\ D_1^{'} \cdot D_2 +_p D_1^{''} \cdot D_2 & D_1 = D_1^{'} +_p D_1^{''} \\ a \cdot (D_1^{'} \circ D_2) & D_1 = a \cdot D_1^{'} \end{cases}$$

For a role B, its expansion to a deterministic process, denoted $\mathsf{exp}(B)$, is defined as follows.

$$\mathsf{exp}(B) = \begin{cases} a & \text{if } B = a \\ a \cdot \mathsf{exp}(B') & \text{if } B = a \cdot B' \\ \mathsf{exp}(B_1) +_p \mathsf{exp}(B_2) & \text{if } B = B_1 +_p B_2 \\ \mathsf{exp}(B_1) \circ \mathsf{exp}(B') +_p & \text{if } B = (B_1 +_p B_2) \cdot B' \\ \mathsf{exp}(B_2) \circ \mathsf{exp}(B') & \end{cases}$$

The set of interleavings for the parallel composition of deterministic processes is given in Figure 5.2, and will be denoted by $\mathcal{I}(P)$ for a process P. Recall that a process Q takes the

form $a_1 \cdot \ldots \cdot a_n \cdot (B_1 \mid \ldots \mid B_m)$ where $a_i \in \{\nu x, (x := u)\}$. If $D \in \mathcal{I}(\mathsf{exp}(B_1) \mid \ldots \mid \mathsf{exp}(B_m))$, then $a_1 \cdot \ldots \cdot a_n \cdot D \in \mathcal{I}(P)$. Notice that $\mathcal{I}(Q)$ is a deterministic process. Further, for any adversary $\mathcal{A}$ and process Q there exists a deterministic process $D \in \mathcal{I}(Q)$ such that $\mathsf{Exec}([\![D]\!]^{\mathcal{A}}) = \mathsf{Exec}([\![Q]\!]^{\mathcal{A}})$ modulo a renaming of variables. We will write $\mathcal{I}(Q, \mathcal{A})$ to denote such a deterministic process. For a bitstring ω, if $a \in \{\nu x, \mathtt{in}(x), (x := u)\}$, let a^ω be νx^ω, $\mathtt{in}(x^\omega)$ or $(x^\omega := u)$, respectively. For any deterministic process D and bitstring ω, let D^ω be

$$D^\omega = \begin{cases} a & \text{if } D = a \text{ and } a \notin \overline{a} \\ a^\omega & \text{if } D = a \text{ and } a \in \overline{a} \\ a \cdot D_0^\omega & \text{if } D = a \cdot D_0 \text{ and } a \notin \overline{a} \\ a^\omega \cdot D_0^\omega\{x \mapsto x^\omega\} & \text{if } D = a \cdot D_0 \text{ and } a \in \overline{a} \\ D_0^{\omega 0} +_p D_1^{\omega 1} & \text{if } D = D_0 +_p D_1 \end{cases}$$

for $\overline{a} = \{\nu x, \mathtt{in}(x), (x := u)\}$. By construction, D^ω is such that for any adversary $\mathcal{A}$ and $\rho_1, \rho_2 \in \mathsf{Exec}([\![D^\omega]\!]^{\mathcal{A}})$ if $\mathsf{last}(\rho_1) = (D_1, \varphi_1, \sigma_1)$, $\mathsf{last}(\rho_2) = (D_2, \varphi_2, \sigma_2)$ and $x \in \mathsf{dom}(\sigma_1) \cap \mathsf{dom}(\sigma_2)$ then $x\sigma_1 = x\sigma_2$. Let $\rho_1, \ldots, \rho_m = \mathsf{MExec}([\![D^\omega]\!]^{\mathcal{A}})$ where $\mathsf{last}(\rho_j) = (D_j, \varphi_j, \sigma_j)$ for all $j \in \{1, \ldots, m\}$. Define $\mathsf{bind}(D^\omega, \mathcal{A}) = \bigcup_{j=1}^{j=m} \sigma_j$. For any process Q, let $Q^b_{\mathcal{L}_b}$ be the result of replacing, in Q, every occurrence of a variable x that occurs in an atomic process a^ℓ, where $\ell \in \mathcal{L}_b$, by the variable x^b. For the remainder of this section, let $i \in \{1, \ldots, n\}$, $S = \{x_1, \ldots, x_n\}$ and

$$C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid \ldots \mid D_n[\Box_n])$$

$$C'[\Box_1, \ldots, \Box_n] = \nu k'_1 \cdot \ldots \cdot \nu k'_m \cdot (D'_1[\Box_1] \mid D'_2[\Box_2] \mid \ldots \mid D'_n[\Box_n])$$

be contexts over $\mathcal{F}_c$ with labels from $\mathcal{L}_c$, $B_1, \ldots, B_n$ (resp. $B'_1, \ldots, B'_n$) be roles over $\mathcal{F}_b$ with labels from $\mathcal{L}_b$ and assume that the conditions of Theorem 5.1 hold. Let $\mathcal{A}$ be an adversary for $C[B_1, \ldots, B_n]$ and $C'[B'_1, \ldots, B'_n]$. We will henceforth refer to the preceding conditions by †. By convention, the labels from each B_i come from different equivalence classes, which we will denote by $\sim_i$. Define

$$S_D = \{x^\omega \mid x^\omega \in \mathsf{vars}(\mathcal{I}(C[B_1, \ldots, B_n], \mathcal{A})^\epsilon) \wedge x \in \mathsf{vars}(S)\}$$

and

$$S'_D = \{x^\omega \mid x^\omega \in \mathsf{vars}(\mathcal{I}(C'[B'_1, \ldots, B'_n], \mathcal{A})^\epsilon) \wedge x \in \mathsf{vars}(S)\}.$$

Let $M = \mathsf{max}(|S_D|, |S'_D|)$.

Definition 5.6 *Define $\Delta(C[B_1, \ldots, B_n], \mathcal{A}, M)$ to be the process*

$$[T]_1 \cdot \ldots \cdot [T]_s \cdot \nu k'_1 \cdot \ldots \cdot \nu k'_v \cdot (y^b_1 := z_1) \cdot \ldots \cdot (y^b_v := z_v) \cdot (\mathcal{I}(C[B_1, \ldots, B_n], \mathcal{A})^\epsilon)^b_{\mathcal{L}_b}$$

such that $\{y_1, \ldots, y_v\} = S_D$, $z_j \in \{k'_1, \ldots, k'_v\}$ *and for all* z_j, z_h *we have* $z_j = z_h$ *iff* $\sigma^{\mathcal{A}}(y_j) =_E \sigma^{\mathcal{A}}(y_h)$ *for* $\sigma^{\mathcal{A}} = \mathsf{bind}(C[B_1, \ldots, B_n], \mathcal{A})$. *We also require* s *is such that* $s + 2v = M$.

In Definition 5.6, every atomic action in the prefix $[T]_1 \cdot \ldots \cdot [T]_s \cdot \nu k'_1 \cdot \ldots \cdot \nu k'_v \cdot (y^b_1 := z_1) \cdot \ldots \cdot (y^b_v := z_v)$ is labeled sequentially with $\ell_1, \ldots, \ell_{2v+s} \in \mathcal{L}_b$. We can similarly define $\Delta(C'[B'_1, \ldots, B'_n], \mathcal{A}, M)$. Let $\mathsf{dis}^{\mathcal{A}} : \mathsf{A} \to \mathsf{A}$ be the adversary such that if $\mathcal{A}'(\overline{o}) = \alpha$ then $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}')(\overline{o}_0\overline{o}) = \alpha$ where $\overline{o}_0 = o_0(\tau, [\ell_1]) \ldots (\tau, [\ell_{2v+s}])o_{2v+s}$. In what follows, when we write $\rho \vdash S$ for some $\rho \in \mathsf{Exec}([\![P]\!])$ and $S \subseteq \mathsf{vars}(P)$ we mean that there exists an $x \in S$ such $\varphi \vdash x\sigma$ where $\mathsf{last}(\rho) = (P', \varphi, \sigma)$. We will also assume that $B_1, \ldots, B_n$ and $C[\Box_1, \ldots, \Box_n]$ only share the variables $S = \{x_1, \ldots, x_n\}$. We will write $\mathsf{vars}^b(C[B_1, \ldots, B_n])$ and $\mathsf{vars}^c(C[B_1, \ldots, B_n])$ to denote the sets $\bigcup_{i=1}^n \mathsf{vars}(B_i) \setminus x_i$ and $\mathsf{vars}(C[\Box_1, \ldots, \Box_n]) \setminus S$ respectively.

Lemma 5.14 *Let* $\mathcal{A}$ *be an adversary for* $P_0 = C[B_1, \ldots, B_n]$ *and* $P_1 = C'[B'_1, \ldots, B'_n]$. *Assuming* †, *If* $Q_0 = \Delta(P_0, \mathcal{A}, M)$ *and* $Q_1 = \Delta(P_1, \mathcal{A}, M)$ *then* (P_0, P_1) *is transposble to* (Q_0, Q_1) *by* $\mathsf{dis}^{\mathcal{A}}$.

Proof. Observe that $\mathsf{Exec}([\![P_0]\!]^{\mathcal{A}}) = \mathsf{Exec}([\![\mathcal{I}(P_0, \mathcal{A})]\!]^{\mathcal{A}})$ and $\mathcal{I}(P_0, \mathcal{A})$ is the same as $\mathcal{I}(P_0, \mathcal{A})^\epsilon$ modulo renaming of variables. Likewise, $\mathsf{Exec}([\![P_1]\!]^{\mathcal{A}}) = \mathsf{Exec}([\![\mathcal{I}(P_1, \mathcal{A})]\!]^{\mathcal{A}})$ and $\mathcal{I}(P_1, \mathcal{A})$ is the same as $\mathcal{I}(P_1, \mathcal{A})^\epsilon$ modulo renaming of variables. Thus, it suffices to show the pair $(\mathcal{I}(P_0, \mathcal{A})^\epsilon, \mathcal{I}(P_0, \mathcal{A})^\epsilon)$ is transposable to (Q_0, Q_1) by $\mathsf{dis}^{\mathcal{A}}$. We define a bijection $\delta : \mathsf{MExec}([\![\mathcal{I}(P_0, \mathcal{A})^\epsilon]\!]^{\mathcal{A}}) \uplus \mathsf{MExec}([\![\mathcal{I}(P_1, \mathcal{A})^\epsilon]\!]^{\mathcal{A}}) \to \mathsf{MExec}([\![Q_0]\!]^{\mathsf{dis}^{\mathcal{A}}}) \uplus \mathsf{MExec}([\![Q_1]\!]^{\mathsf{dis}^{\mathcal{A}}})$. The definition of δ is only given on executions of $\mathsf{MExec}([\![\mathcal{I}(P_0, \mathcal{A})^\epsilon]\!]^{\mathcal{A}})$, it can be naturally extended to executions of $\mathsf{MExec}([\![\mathcal{I}(P_1, \mathcal{A})^\epsilon]\!]^{\mathcal{A}})$. Let $\mathsf{dis}^{\mathcal{A}}(\mathcal{A}) = \mathcal{A}'$ and $P'_0 = \mathcal{I}(P_0, \mathcal{A})^\epsilon$. Define a bijection

$$\delta_0 : \mathsf{MExec}([\![P'_0]\!]^{\mathcal{A}}) \to \mathsf{MExec}([\![Q_0]\!]^{\mathcal{A}'})$$

as follows. To begin, let $\rho_0 = (Q_0, \emptyset, \emptyset) \xrightarrow{(\tau,[\ell])} \ldots \xrightarrow{(\tau,[\ell])} ((R_0)^b_{\mathcal{L}_b}, \emptyset, \sigma_0)$ for $\ell \in \mathcal{L}_b$ and $\mathsf{dom}(\sigma) = \{k'_1, \ldots, k'_v, y^b_1, \ldots, y^b_v\}$ be an execution of $\mathsf{MExec}([\![Q_0]\!]^{\mathcal{A}'})$ such that $|\rho_0| = M$. Now consider any $\rho \in \mathsf{MExec}([\![P'_0]\!]^{\mathcal{A}})$ of the form $(R_0, \varphi_0, \sigma_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} (R_m, \varphi_m, \sigma_m)$. Define $\rho' = ((R_0)^b_{\mathcal{L}_b}, \varphi'_0, \sigma'_0) \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_m} ((R_m)^b_{\mathcal{L}_b}, \varphi'_m, \sigma'_m)$ where for all $j \in \{1, \ldots, m\}$, φ'_j and σ'_j are as follows. If $\sigma^{\mathcal{A}} = \mathsf{bind}(P_1, \mathcal{A})$, $\tilde{s} = \{\sigma^{\mathcal{A}}(y_1), \ldots, \sigma^{\mathcal{A}}(y_v)\}$ and $\tilde{n} = \{\sigma_0(y^b_1), \ldots, \sigma_0(y^b_v)\}$ then σ'_j and φ'_j are smallest substitutions that satisfy the following.

1. $\sigma'_j(x) = \sigma_0(x)$ for all $x \in \mathsf{dom}(\sigma_0)$
2. $\sigma'_j(x^b) = R^{\tilde{n}}_{\tilde{s},b}(\mathsf{col}(\sigma_j(x^b)))$ for all $x^b \in \mathsf{dom}(\sigma'_j) \cap \mathsf{vars}^b(P'_0)$

3. $\sigma'_j(x) = R^{\tilde{n}}_{\tilde{s},b}(\mathsf{col}(\sigma_j(x)))$ for all $x \in \mathsf{dom}(\sigma'_j) \cap \mathsf{vars}^c(P'_0)$

4. $\varphi'_j(w_{i,[\ell]}) = R^{\tilde{n}}_{\tilde{s},b}(\mathsf{col}(\varphi_j(w_{i,[\ell]})))$

Define $\delta_0(\rho) = \rho_0\rho'$. Let $X_D = \{x^\omega | x^\omega \in \mathsf{vars}(P'_0) \text{ and } x \in S\}$ and $X'_D = X_D \cup \{(x^\omega)^b | (x^\omega)^b \in \mathsf{vars}(Q_0) \text{ and } x \in S\}$. Consider any $\rho \in \mathsf{MExec}([\![P'_0]\!]^{\mathcal{A}})$. By condition 4 of Theorem 5.1 $\rho \not\models \mathsf{secret}(X_D)$. Applying Proposition 5.2, the linear process $L(\rho) \not\models \mathsf{secret}(X_D)$. Furthermore, $L(\rho)$ and $L(\delta_0(\rho))$ meet the conditions of Theorem 1 from [54], allowing one to conclude that $L(\delta_0(\rho)) \not\models \mathsf{secret}(X'_D)$ and $\delta_0(\rho) \in \mathsf{MExec}([\![Q_0]\!]^{\mathcal{A}'})$. Proposition 5.2 again yields $\delta_0(\rho) \not\models \mathsf{secret}(X'_D)$. In particular, this means that $\rho \not\models \tilde{s}$ and $\delta_0(\rho) \not\models \tilde{s}$ and we can apply Lemma 5.7 to conclude $\varphi \equiv \varphi'$ and hence $\mathsf{obs}(\mathsf{last}(\rho)) = \mathsf{obs}(\mathsf{last}(\delta_0(\rho)))$. As noted previously, we can similarly define a bijection $\delta_1 : \mathsf{MExec}([\![\mathcal{I}(P_1, \mathcal{A})^\epsilon]\!]^{\mathcal{A}}) \to \mathsf{MExec}([\![Q_1]\!]^{\mathcal{A}'})$ that satisfies the properties above. Let $\delta = \delta_0 \uplus \delta_1$. $\mathcal{A}'$ is an adversary for Q_0 and Q_1 such that property 2 of Definition 5.5 holds. Clearly property 1 also holds by the definition of δ.

Lemma 5.15 *Assuming †, if* $C[B_1, \ldots, B_n] \not\approx C'[B'_1, \ldots, B'_n]$ *then*

$$C[\mathtt{out}(\sharp(x_1)) \cdot B_1, \ldots, \mathtt{out}(\sharp(x_n)) \cdot B_n] \not\approx C'[\mathtt{out}(\sharp(x_1)) \cdot B'_1, \ldots, \mathtt{out}(\sharp(x_n)) \cdot B'_n].$$

Proof. Let $P_1 = C[B_1, \ldots, B_n]$, $P_2 = C'[B'_1, \ldots, B'_n]$, $P'_1 = C[\mathtt{out}(\sharp(x_1)) \cdot B_1, \ldots, \mathtt{out}(\sharp(x_n)) \cdot B_n]$ and $P'_2 = C'[\mathtt{out}(\sharp(x_1)) \cdot B'_1, \ldots, \mathtt{out}(\sharp(x_n)) \cdot B'_n]$. We will assume $\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))$ are labeled by $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$. Define a projection π from executions of P'_1 (resp. P'_2) to executions of P_1 (resp. P_2) inductively as follows. If $\rho \in \mathsf{Exec}([\![P'_1]\!])$ (resp. $\rho \in \mathsf{Exec}([\![P'_2]\!])$), contains no actions, then $\pi(P'_1, \emptyset, \emptyset) = (P_1, \emptyset, \emptyset)$ (resp. $\pi(P'_2, \emptyset, \emptyset) = (P_2, \emptyset, \emptyset)$). Inductively, let $\rho = \rho_0 \xrightarrow{\alpha} z$ for $\alpha = (\S, [\ell])$. If $\ell = \ell_i$ for $i \in \{1, \ldots, n\}$ then $\pi(\rho) = \pi(\rho_0)$. Otherwise $\pi(\rho) = \pi(\rho_0) \xrightarrow{\alpha} z'$. The projection π can be extended to traces in the following way. For a trace $\overline{o}$, $\pi(\overline{o}) = \mathsf{tr}(\pi(\rho'))$ where ρ' is any execution such that $\mathsf{tr}(\rho') = \overline{o}$. This extension is well defined. From π we can define an adversary $\mathcal{A}'$ for P'_1 (resp. P'_2) from an adversary $\mathcal{A}$ for P_1 (resp. P_2) in the following way. For a trace $\overline{o}$, if $\mathcal{A}(\pi(\overline{o})) = (\S, [\ell])$ where $\ell \in \sim_i$ and $\overline{o}$ doesn't contain the action $(\tau, [\ell_i])$, then $\mathcal{A}'(\overline{o}) = (\tau, [\ell_i])$. Otherwise $\mathcal{A}'(\overline{o}) = \mathcal{A}(\pi(\overline{o}))$. Let $\rho, \rho_1, \rho_2 \in \mathsf{MExec}([\![P'_k]\!]^{\mathcal{A}'})$ for $k \in \{1, 2\}$. First observe that $\mathsf{prob}_{P'_k}(\rho, \mathcal{A}') = \mathsf{prob}_{P_k}(\pi(\rho), \mathcal{A})$ by our definition of $\mathcal{A}'$. Furthermore, if $\mathsf{tr}(\pi(\rho_1)) \neq \mathsf{tr}(\pi(\rho_2))$ then $\mathsf{tr}(\rho_1) \neq \mathsf{tr}(\rho_2)$. Because $P_1 \not\approx P_2$, by Proposition 5.1, there is an adversary $\mathcal{A}$ and trace $\overline{o}$ that is maximal for $[\![P_1]\!]^{\mathcal{A}}$ and $[\![P_2]\!]^{\mathcal{A}}$ such that $\mathsf{prob}_{P_1}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{P_2}(\overline{o}, \mathcal{A})$. Let $\overline{o}_1, \ldots, \overline{o}_k$ be the traces of $[\![P'_1]\!]^{\mathcal{A}'}$ and $[\![P'_2]\!]^{\mathcal{A}'}$ such that $\pi(\overline{o}_j) = \overline{o}$ for all $j \in \{1, \ldots, k\}$. We have

$$\mathsf{prob}_{P_1}(\overline{o}, \mathcal{A}) = \sum_{j=1}^{k} \mathsf{prob}_{P_1}(\overline{o}_j, \mathcal{A}') \quad \text{and} \quad \mathsf{prob}_{P_2}(\overline{o}, \mathcal{A}) = \sum_{j=1}^{k} \mathsf{prob}_{P'_2}(\overline{o}_j, \mathcal{A}').$$

By the preceding observations,

$$\sum_{j=1}^{k} \mathsf{prob}_{P_1'}(\overline{o}_j, \mathcal{A}') \neq \sum_{j=1}^{k} \mathsf{prob}_{P_2'}(\overline{o}_j, \mathcal{A}')$$

and thus there exists j such that $\mathsf{prob}_{P_1'}(\overline{o}_j, \mathcal{A}') \neq \mathsf{prob}_{P_2'}(\overline{o}_j, \mathcal{A}')$. That is, $P_1' \not\approx P_2'$.

Lemma 5.16 *Let P, P', Q and Q' be processes such that $\mathsf{vars}(P) \cap \mathsf{vars}(Q) = \{x\}$ and $\mathsf{vars}(P') \cap \mathsf{vars}(Q') = \{x\}$. If $\nu x \cdot (P \mid Q) \approx \nu x \cdot (P' \mid Q')$ then*

$$\nu x_1 \cdot P\{x \mapsto x_1\} \mid \nu x_2 \cdot Q\{x \mapsto x_2\} \approx \nu x_1 \cdot P'\{x \mapsto x_1\} \mid \nu x_2 \cdot Q'\{x \mapsto x_2\}.$$

Proof. We begin by showing that $\nu x \cdot P \approx \nu x \cdot P'$ and $\nu x \cdot Q \approx \nu x \cdot Q'$. We only give the agruement for $\nu x \cdot P \approx \nu x \cdot P'$ as the case of $\nu x \cdot Q \approx \nu x \cdot Q'$ is similar. Assume for a contradiction that $\nu x \cdot P \not\approx \nu x \cdot P'$. There exists an adversary $\mathcal{A}$ and trace $\overline{o}$ such that $\mathsf{prob}_{\nu x \cdot P}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{\nu x \cdot P'}(\overline{o}, \mathcal{A})$. Clearly, $\mathcal{A}$ is also an adversary for $\nu x \cdot (P|Q)$ and $\nu x \cdot (P'|Q')$ such that $\mathsf{prob}_{\nu x \cdot P}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{\nu x \cdot (P|Q)}(\overline{o}, \mathcal{A})$ and $\mathsf{prob}_{\nu x \cdot P'}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{\nu x \cdot (P'|Q')}(\overline{o}, \mathcal{A})$. That is, $\nu x \cdot (P \mid Q) \approx \nu x \cdot (P' \mid Q')$, contradiction. Given that $\nu x \cdot P \approx \nu x \cdot P'$ and $\nu x \cdot Q \approx \nu x \cdot Q'$ the result is a consequence of Lemma 5.1.

Lemma 5.17 *Assuming †, we have $C[B_1, ..., B_n] \approx C'[B_1, ..., B_n]$.*

Proof. Assume for a contradiction that $C[B_1, \ldots, B_n] \not\approx C'[B_1, \ldots, B_n]$. Let

$$P_1 = C[\mathtt{out}(\sharp(x_1)) \cdot B_1, \ldots, \mathtt{out}(\sharp(x_n)) \cdot B_n]$$

and

$$P_2 = C'[\mathtt{out}(\sharp(x_1)) \cdot B_1, \ldots, \mathtt{out}(\sharp(x_n)) \cdot B_n].$$

By Lemma 5.15, $P_1 \not\approx P_2$. Let $\mathcal{A}$ be an adversary such that $\mathsf{prob}_{P_1}(\overline{o}, \mathcal{A}) \neq \mathsf{prob}_{P_2}(\overline{o}, \mathcal{A})$ for some trace $\overline{o}$. By Lemma 5.14, there exists some $M \in \mathbb{N}$ such that (P_1, P_2) are transposable to $(\Delta(P_1, \mathcal{A}, M), \Delta(P_2, \mathcal{A}, M))$. By Lemma 5.13, this yields $\Delta(P_1, \mathcal{A}, M) \not\approx \Delta(P_2, \mathcal{A}, M)$ and again we have a trace $\overline{o}_0$ and an adversary $\mathcal{A}'$ such that

$$\mathsf{prob}_{\Delta(P_1, \mathcal{A}, M)}(\overline{o}_0, \mathcal{A}') \neq \mathsf{prob}_{\Delta(P_2, \mathcal{A}, M)}(\overline{o}_0, \mathcal{A}').$$

Further, let $\rho_1, \rho_2 \in \mathsf{Exec}([\![\Delta(P_1, \mathcal{A}, M)]\!]^{\mathcal{A}'})$ be such that $\mathsf{tr}(\rho_1) = \mathsf{tr}(\rho_2) = \overline{o}_0$. If $\overline{o}_0$ contains an action with a label from $\sim_i$, then the first such action in ρ_1 (resp. ρ_2) is an output of a

term of the form $\sharp((x_i^{\pi_1})^b\sigma_1)$ (resp. $\sharp((x_i^{\pi_2})^b\sigma_2)$) where $\mathsf{last}(\rho_k) = (R_k, \varphi_k, \sigma_k)$ for $k \in \{1,2\}$. We will write $\sharp^1(x_i)$ (resp. $\sharp^2(x_i)$) to denote the first (and only) output of ρ_1 (resp. ρ_2) with a label from $\sim_i$. Observe that if ρ_1 contains actions with labels from $\sim_i$ and $\sim_j$ then $\sharp^1(x_i) =_E \sharp^1(x_j)$ iff $\sharp^2(x_i) =_E \sharp^2(x_j)$. If this was not the case, then ρ_1 and ρ_2 would not have the same trace. Notice that the above observation also holds when $\rho_2 \in \mathsf{Exec}([\![\Delta(P_2, \mathcal{A}, M)]\!]^{\mathcal{A}'})$.

Let $B_0 := \nu k_0 \cdot \ldots \cdot \nu k_n \cdot (x_1 := z_1) \cdot \ldots \cdot (x_1 := z_n)$ be a process such that $z_i \in \{k_0, \ldots, k_n\}$ and the following hold. If $\overline{o}_0$ doesn't contain an action from $\sim_i$ then $z_i = k_0$ and otherwise $z_i = z_j$ iff $\sharp^1(x_i) = \sharp^1(x_j)$. By definition, $\Delta(P_1, \mathcal{A}, M)$ and $\Delta(P_2, \mathcal{A}, M)$, both contain a prefix of the form $\nu k'_1 \cdot \ldots \cdot \nu k'_v \cdot (y_1^b := z_1) \cdot \ldots \cdot (y_v^b := z_v)$. Furthermore, each prefix has the same length. Let $B'_0 = [\top] \cdot \ldots \cdot [\top] \cdot B_0$ such that $|B'_0| = 2v$. We will assume that the actions of B'_0 are labeled sequentially by $\ell_1, \ldots, \ell_{2v} \in \mathcal{L}_b$. Using the adversary $\mathcal{A}'$ and the trace $\overline{o}_0$, we construct an adversary $\mathcal{A}''$ for $C[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \mid B'$ and $C'[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \mid B'$ where $B' = B'_0 \cdot (B_1 \mid \ldots \mid B_n)$ as follows. If $\overline{o}'_0$ is a prefix of $\overline{o}_0$ then $\mathcal{A}''(\overline{o}'_0) = \mathcal{A}'(\overline{o}'_0)$. Otherwise, $\mathcal{A}''$ is undefined. By our construction, we have

$$\mathsf{prob}_{(C[\mathtt{out}(\sharp(x_1)),\ldots,\mathtt{out}(\sharp(x_n))]\mid B')}(\overline{o}_0, \mathcal{A}'') \neq \mathsf{prob}_{(C'[\mathtt{out}(\sharp(x_1)),\ldots,\mathtt{out}(\sharp(x_n))]\mid B')}(\overline{o}_0, \mathcal{A}'')$$

which means

$$C[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \mid B' \not\approx C'[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \mid B'.$$

Applying Lemma 5.12, we get

$$C[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))] \not\approx C'[\mathtt{out}(\sharp(x_1)), \ldots, \mathtt{out}(\sharp(x_n))]$$

which contradicts condition 5 of Theorem 5.1.

Lemma 5.18 *Assuming †, we have* $C'[B_1, \ldots, B_n] \approx C'[B'_1, \ldots, B'_n]$.

Proof. By a similar arguement as the one used in Lemma 5.17, there exists a process $B_0 := \nu k_0 \cdot \ldots \cdot \nu k_n \cdot (x_1 := z_1) \cdot \ldots \cdot (x_1 := z_n)$ where $z_i \in \{k_0, \ldots, k_n\}$ for all $i \in \{1, \ldots, n\}$ such that

$$B_0 \cdot (B_1 \mid \ldots \mid B_n) \not\approx B_0 \cdot (B'_1 \mid \ldots \mid B'_n).$$

By Lemma 5.16, we can conclude that

$$\begin{gathered} \nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \\ \not\approx \\ \nu k \cdot (x'_1 := k) \cdot \ldots \cdot (x'_n := k) \cdot (B'_1 \mid \ldots \mid B'_n) \end{gathered}$$

which contradictions condition 6 of Theorem 5.1.

Theorem 5.1 is a consequence of Lemma 5.17 and Lemma 5.18.

5.6 SHARED PRIMITIVES THROUGH TAGGING

Theorem 5.1 requires that the context and basic processes don't share cryptographic primitives. To extend the result to processes that allow components of the composition to share primitives, such as functions for encryption, decryption and hashing, we utilize a syntactic transformation of a protocol and its signature called tagging. When a protocol is tagged, a special identifier is appended to each of the messages that it outputs. On input, the protocol recursively tests all subterms of the input message to verify their tags are consistent with the protocol's tag. If this requirement is not met, the protocol deadlocks. In Theorem 5.2, we show that an attack on a composition of two tagged protocols originating from the same signature can be mapped to an attack on the composition of the protocols when the signatures are explicitly made disjoint. In this result, we consider the fixed equational theory $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ from Example 2.1. For this theory, we define a signature renaming function $\_^d$ which transforms a context C over the signature $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$ to a context C^d by replacing every occurrence of the function symbols $\mathsf{senc}, \mathsf{sdec}$ and h in C by $\mathsf{senc}_d, \mathsf{sdec}_d$ and h_d, respectively. Given a context $C[\Box_1, \ldots, \Box_n]$ and basic processes $B_1, \ldots, B_n$ we write $\lceil C[B_1, \ldots, B_n] \rceil$ to denote the tagged version of $C[B_1, \ldots, B_n]$ (see Definition 5.7). We now give our composition result for protocols over a common equational theory.

Theorem 5.2 *Let* $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot (D_1[\Box_1] \mid \ldots \mid D_n[\Box_n])$ *(resp.* $C'[\Box_1, \ldots, \Box_n] = \nu k'_1 \cdot \ldots \cdot \nu k'_m \cdot (D'_1[\Box_1] \mid \ldots \mid D'_n[\Box_n])$*) be a context over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_c$*. Further let* $B_1, \ldots, B_n$ *(resp.* $B'_1, \ldots, B'_n$*) be roles over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_b$*. For* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$ *and* $\sharp \notin \mathcal{F}_b \cup \mathcal{F}_c$*, assume that the following hold.*

1. $\mathsf{fv}(C) = \mathsf{fv}(C') = \emptyset$, $\mathsf{fv}(B_i) = \{x_i\}$ *and* $\mathsf{fv}(B'_i) = \{x'_i\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *and* $\mathsf{vars}(C') \cap \mathsf{vars}(B'_i) = \{x'_i\}$
3. $C[B_1, \ldots, B_n]$ *and* $C'[B'_1, \ldots, B'_n]$ *are ground*
4. $C[B_1, \ldots, B_n] \models_{E_{\mathsf{senc}},1} \mathsf{secret}(x_1, \ldots, x_n)$ *and* $C'[B'_1, \ldots, B'_n] \models_{E_{\mathsf{senc}},1} \mathsf{secret}(x'_1, \ldots, x'_n)$
5. $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \approx C'[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}]$
6. $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \approx \nu k \cdot (x'_1 := k) \cdot \ldots \cdot (x'_n := k) \cdot (B'_1 \mid \ldots \mid B'_n)$

Then $\lceil C^c[B_1^b, \ldots, B_n^b] \rceil \approx \lceil (C')^c[(B_1')^b, \ldots, (B_n')^b] \rceil$.

The remainder of this section is dedicated to the proof of Theorem 5.2. Let C be a context and B be a basic process, both over the equational theory $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$. To securely compose C and B, the terms occurring in each protocol must be tagged by function symbols from disjoint equational theories. The tagging of two protocols will be done in two steps. To begin, the signature renaming function $\_^d$ will be applied to each of C and B with distinct values of $d \in \{b, c\}$. The resulting context C^d is over the signature $(\mathcal{F}^d_{\mathsf{senc}}, E^d_{\mathsf{senc}})$, for $\mathcal{F}^d_{\mathsf{senc}} = \{\mathsf{senc}_d, \mathsf{sdec}_d, \mathsf{h}_d\}$ and $E^d_{\mathsf{senc}} = \{\mathsf{sdec}_d(\mathsf{senc}_d(m, k), k) = m\}$. Given C^c and B^b over the disjoint signatures $\mathcal{F}^c_{\mathsf{senc}}$ and $\mathcal{F}^b_{\mathsf{senc}}$, the tagging function $\lceil \_ \rceil$ is then applied to C^c and B^b, generating the the tagged versions of C and B. We now give the formal definition of the tagging function $\lceil \_ \rceil$. Let $\mathcal{F}^d_{\mathsf{tag}} = \{\mathsf{tag}_d, \mathsf{untag}_d\}$ and $E^d_{\mathsf{tag}} = \{\mathsf{untag}_d(\mathsf{tag}_d(x)) = x\}$. Further, $\mathcal{F}_{\mathsf{tag}} = \mathcal{F}^b_{\mathsf{tag}} \cup \mathcal{F}^c_{\mathsf{tag}}$ and $E_{\mathsf{tag}} = E^b_{\mathsf{tag}} \cup E^c_{\mathsf{tag}}$. The function $\mathcal{H} : \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}^d_{\mathsf{tag}}, \mathcal{X})$ is defined below.

$$
\begin{array}{lcl}
\mathcal{H}(\mathsf{senc}_d(u_1, u_2)) & = & \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)), \mathcal{H}(u_2)) \\
\mathcal{H}(\mathsf{sdec}_d(u_1, u_2)) & = & \mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1), \mathcal{H}(u_2))) \\
\mathcal{H}(\mathsf{h}_d(u)) & = & \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u))) \\
\mathcal{H}(u) & = & u, \text{ if } u \text{ is a name or variable}
\end{array}
$$

The function tests^d below maps terms from $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}^d_{\mathsf{tag}}, \mathcal{X})$ to a conjunction of equalities, as defined below.

$$
\begin{array}{lcl}
\mathsf{tests}^d(\mathsf{senc}(u_1, u_2)) & = & \mathsf{tests}^d(u_1) \wedge \mathsf{tests}^d(u_2) \\
\mathsf{tests}^d(\mathsf{sdec}(u_1, u_2)) & = & \mathsf{tests}^d(u_1) \wedge \mathsf{tests}^d(u_2) \\
\mathsf{tests}^d(\mathsf{h}(u)) & = & \mathsf{tests}^d(u) \\
\mathsf{tests}^d(\mathsf{tag}_d(u)) & = & \mathsf{tests}^d(u) \\
\mathsf{tests}^d(\mathsf{untag}_d(u)) & = & \mathsf{tag}_d(\mathsf{untag}_d(u)) = \mathsf{tag}_d(u) \wedge \\
 & & \mathsf{tests}^d(u) \\
\mathsf{tests}^d(u) & = & \top, \text{ if } u \text{ is a name or variable}
\end{array}
$$

For a term u, observe that $\mathsf{tests}^d(u) = c_1 \wedge \ldots \wedge c_n$ where c_i is $\top$ or $v_1 = v_2$ for ground terms $v_1, v_2 \in \mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}$. We say that $\mathsf{tests}^d(u)$ *passes* if c_i is $\top$ or $v_1 =_{E_{\mathsf{senc}} \cup E_{\mathsf{tag}}} v_2$ for all $i \in \{1, \ldots, n\}$. Using the preceding notions, we define $\lceil \_ \rceil$ as follows.

Definition 5.7 *Let B^d be basic process over $\mathcal{F}^d_{\mathsf{senc}}$ for $d \in \{b, c\}$. The basic process $\lceil B^d \rceil$ is defined as follows.*

$$
\begin{array}{lcl}
\lceil \Box \rceil & = & \Box \\
\lceil \nu x \rceil & = & [\top] \cdot \nu x \\
\lceil \mathtt{in}(x) \rceil & = & [\top] \cdot \mathtt{in}(x) \\
\lceil \mathtt{out}(u) \rceil & = & [\mathsf{tests}^d(\mathcal{H}(u))] \cdot \mathtt{out}(\mathcal{H}(u)) \\
\lceil (x := u) \rceil & = & [\mathsf{tests}^d(\mathcal{H}(u))] \cdot (x := \mathcal{H}(u)) \\
\lceil [u = v] \rceil & = & [\mathsf{tests}^d(\mathcal{H}(u)) \wedge \mathsf{tests}^d(\mathcal{H}(v))] \cdot \\
 & & [\mathcal{H}(u) = \mathcal{H}(v)] \\
\lceil B_1 \cdot B_2 \rceil & = & \lceil B_1 \rceil \cdot \lceil B_2 \rceil \\
\lceil B_1 +_p B_2 \rceil & = & [\top] \cdot \lceil B_1 \rceil +_p [\top] \cdot \lceil B_2 \rceil
\end{array}
$$

Definition 5.7 can be lifted naturally to basic contexts by requiring $\lceil \Box \rceil = \Box$ for any process variable $\Box$.

Definition 5.8 *Let $C^d = a_1 \cdot \ldots \cdot a_n \cdot (D_1[\Box_1] \mid \ldots \mid D_n(\Box_n))$ be a context over $\mathcal{F}^d_{\mathsf{senc}}$ for $d \in \{b, c\}$. The context $\lceil C^d \rceil$ is $\lceil a_1 \cdot \ldots \cdot a_n \rceil \cdot (\lceil D_1[\Box_1] \rceil \mid \ldots \mid \lceil D_n(\Box_n) \rceil)$.*

The following example demonstrates the behavior of processes that are tagged using our scheme. Whenever a protocol manipulates a term, that term should be tagged with the identifier of the protocol. To enforce this, every observable action in a tagged protocol is prefixed with a conjunction of tests. If the terms manipulated by the atomic action meet the aforementioned requirement, the tests will pass. Otherwise, the tests will fail, and further protocol actions will be blocked. In this way, messages from one protocol cannot be confused with messages from another protocol.

Example 5.9 *Let $P = \nu n \cdot \nu m \cdot \mathtt{out}(\mathsf{senc}(m, n, k))$ and $Q = \mathtt{in}(x) \cdot \mathtt{out}(\mathsf{sdec}(x, k))$ where the processes P and Q share a key k. We have*

$$\lceil P^b \rceil = [\top] \cdot \nu n \cdot [\top] \cdot \nu m \cdot [\top] \cdot \mathtt{out}(\mathsf{senc}(\mathsf{tag}_b(m), n, k))$$

and

$$\lceil Q^c \rceil = [\top] \cdot \mathtt{in}(x) \cdot [\mathsf{tag}_c(\mathsf{untag}_c(\mathsf{sdec}(x, k))) = \mathsf{sdec}(x, k)] \cdot \mathtt{out}(\mathsf{untag}_c(\mathsf{sdec}(x, k))).$$

If the output of $\lceil P^b \rceil$ is forwarded to $\lceil Q^c \rceil$ then the test

$$\mathsf{tag}_c(\mathsf{untag}_c(\mathsf{sdec}(\mathsf{senc}(\mathsf{tag}_b(m), n, k), k)) = \mathsf{sdec}(\mathsf{senc}(\mathsf{tag}_b(m), n, k), k)$$

reduces to $\mathsf{tag}_c(\mathsf{untag}_c(\mathsf{tag}_b(m)) = \mathsf{tag}_b(m)$ but ultimately blocks because $c \neq b$.

For ease of notation, let $E = E_{\mathsf{senc}} \cup E_{\mathsf{tag}}$. Let $d \in \{b, c\}$ be a symbolic identifier used for tagging processes. We will write d' to denote the new identifier c' if $d = c$ and b' if $d = b$. Using these identifiers, we define new equational theories for tagged processes as follows. Let

$$\mathcal{F}_{\mathsf{tag}}^{d'} = \{\mathsf{tag}_{d'}, \mathsf{untag}_{d'}\}$$

and

$$E_{\mathsf{tag}}^{d'} = \{\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(x)) = x\}.$$

To achieve Theorem 5.2, we will map an attack on a tagged process over the equational theory E to an attack on a process over disjoint signatures in the extended equational theory E_0 defined below. Let

$$\mathcal{F}^0 = \mathcal{F}_{\mathsf{senc}}^b \cup \mathcal{F}_{\mathsf{senc}}^c \cup \mathcal{F}_{\mathsf{senc}}^a \cup \mathcal{F}_{\mathsf{tag}}^{b'} \cup \mathcal{F}_{\mathsf{tag}}^{c'}$$

and

$$E^0 = E_{\mathsf{senc}}^b \cup E_{\mathsf{senc}}^c \cup E_{\mathsf{senc}}^a \cup E_{\mathsf{tag}}^{b'} \cup E_{\mathsf{tag}}^{c'}.$$

We now define a function $\lfloor \_ \rfloor : \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}^0, \mathcal{X})$. This function is an adaptation of the one from [54].

$$\begin{array}{lcl}
\lfloor \mathsf{tag}_d(u) \rfloor & = & \mathsf{tag}_{d'}(\lfloor u \rfloor) \\
\lfloor \mathsf{untag}_d(u) \rfloor & = & \mathsf{untag}_{d'}(\lfloor u \rfloor) \\
\lfloor \mathsf{senc}(u_1, u_2) \rfloor & = & \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor) \text{ if} \\
 & & \lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor)) \\
 & = & \mathsf{senc}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor) \text{ otherwise} \\
\lfloor \mathsf{sdec}(u_1, u_2) \rfloor & = & \mathsf{tag}_{d'}(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)) \text{ if} \\
 & & \lfloor u_1 \rfloor =_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor) \\
 & = & \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor) \text{ otherwise} \\
\lfloor \mathsf{h}(u) \rfloor & = & \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u \rfloor)) \text{ if} \\
 & & \lfloor u \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u \rfloor)) \\
 & = & \mathsf{h}_a(\lfloor u \rfloor) \text{ otherwise} \\
\lfloor u \rfloor & = & u \text{ for a name or a variable}
\end{array}$$

Let u be a term and $\overrightarrow{E}$ be an orientation of the equations of E from left to right. We will write $u \to_E v$ to denote that u rewrites to v and $u \to_E^* v$ if u rewrites to v in 0 or more steps. The normal form of u will be denoted $u\!\downarrow$. A term u is in *head normal form* if any sequence of rewrites on u cannot happen at the root.

Lemma 5.19 *Let $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X})$. If $u \to_E v$ then $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.*

Proof. The proof is by induction on the structure of u. For the base case, let u be a name or variable. If $u \to_E v$ then $u = v$ and the goal is immediate. For the induction step, we proceed by cases.

case 1: $u = \mathsf{tag}_d(u_1)$. By the definition of E, u does not rewrite in the root symbol. That is, if $u \to_E v$ then $v = \mathsf{tag}_d(v_1)$ where $u_1 \to_E v_1$. By definition, $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor v \rfloor = \mathsf{tag}_{d'}(\lfloor v_1 \rfloor)$. By the I.H. we have $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$ and the case follows.

case 2: $u = \mathsf{untag}_d(u_1)$. We consider two cases. First assume that $u_1 = \mathsf{tag}_d(u_2)$ and $u \to_E u_2$. We have the following.

$$\begin{array}{rcl} \lfloor u \rfloor & = & \mathsf{untag}_{d'}(\lfloor u_1 \rfloor) \\ & = & \mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor u_2 \rfloor)) \\ & =_{E^0} & \lfloor u_2 \rfloor \end{array}$$

Otherwise $u \to_E v$ where $v = \mathsf{untag}_d(u_1')$. By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor u_1' \rfloor$ and the result follows.

case 3: $u = \mathsf{senc}(u_1, u_2)$. By the definition of E, u does not rewrite in the root symbol. That is, $v = \mathsf{senc}(v_1, v_2)$ where $u_1 \to_E v_1$ and $u_2 = v_2$ (or $u_1 = v_2$ and $u_2 \to_E v_2$, which follows by a similar argument). By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$. We consider two subcases.

In the first, $\lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$. Because $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$, we have $\lfloor v_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor))$. By definition, we know $\lfloor u \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor), \lfloor u_2 \rfloor)$ and thus $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.

In the second case, $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$. Because $\lfloor u_1 \rfloor =_{E^0} \lfloor v_1 \rfloor$, we have $\lfloor v_1 \rfloor \neq_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor))$. That is, $\lfloor u \rfloor = \mathsf{senc}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_a(\lfloor v_1 \rfloor, \lfloor u_2 \rfloor)$ and thus $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.

case 4: $u = \mathsf{sdec}(u_1, u_2)$. We consider two subcases.

subcase 4.1: $u_1 = \mathsf{senc}(u_3, u_2)$ and $u \to_E u_3$. First assume that the following equation holds.

$$\lfloor u_3 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor)) \tag{5.2}$$

Because $u_1 = \mathsf{senc}(u_3, u_2)$, by the I.H. we have

$$\lfloor u_1 \rfloor =_{E^0} \lfloor \mathsf{senc}(u_3, u_2) \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor)$$

where the latter equality follows by equation 5.2. From the proceeding facts, we have $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$

$$\begin{array}{ll} =_{E^0} & \mathsf{senc}_d(\mathsf{sdec}_d(\mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor), \lfloor u_2 \rfloor), \lfloor u_2 \rfloor) \\ =_{E^0} & \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor) \\ =_{E^0} & \lfloor u_1 \rfloor \end{array}$$

The result is a consequence of the following derivation.

$$\begin{array}{rcl} \lfloor u \rfloor & = & \mathsf{tag}_{d'}(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)) \\ & =_{E^0} & \mathsf{tag}_{d'}(\mathsf{sdec}_d(\mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor), \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)) \\ & =_{E^0} & \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_3 \rfloor)) \\ & =_{E^0} & \lfloor u_3 \rfloor \end{array}$$

Now assume that equation 5.2 does not hold. Because $u_1 \to_E \mathsf{senc}(u_3, u_2)$, by the I.H. we have $\lfloor u_1 \rfloor =_{E^0} \lfloor \mathsf{senc}(u_3, u_2) \rfloor = \mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$, where the latter equality follows from the fact that equation 5.2 does not hold. By the definition of E_0, $\mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$ does not rewrite in the root symbol and $\lfloor u_1 \rfloor \downarrow = \mathsf{senc}_a(u_3', u_2')$. We also have

$$\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)) \downarrow = \mathsf{senc}_d(\mathsf{sdec}_d(\mathsf{senc}_a(u_3', u_4'), u_2').$$

Notice that $\lfloor u_1 \rfloor \downarrow$ and $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$ have normal forms of different size. Because E_0 is a convergent rewrite system, this means that

$$\lfloor u_1 \rfloor \neq_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$$

and thus $\lfloor u \rfloor = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$. The case follows by the derivation below.

$$\begin{array}{rcl} \lfloor u \rfloor & = & \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)) \\ & =_{E^0} & \mathsf{sdec}_a(\mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)) \\ & =_{E^0} & \lfloor u_3 \rfloor \end{array}$$

case 4.2: $u \to_E v$ where $v = \mathsf{sdec}(u_1', u_2)$ and $u_1 \to_E u_1'$. The case when $v = \mathsf{sdec}(u_1, u_2')$ and $u_2 \to_E u_2'$ follows by a similar argument. By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1' \rfloor$ and the case is straightforward.

case 5: $u = \mathsf{h}(u_1)$. Follows by a similar argument as case 2.

Lemma 5.20 *Let $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X})$ be terms in normal form. Then $u \neq v$ implies $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$.*

Proof. Define a function $\lfloor \_ \rfloor_1 : \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}) \to \mathcal{T}(\mathcal{F}^0, \mathcal{X})$ which is identical to $\lfloor \_ \rfloor$ on all cases with the exception that $\lfloor \mathsf{sdec}(u_1, u_2) \rfloor_1 = \mathsf{sdec}_a(\lfloor u_1 \rfloor_1, \lfloor u_2 \rfloor_1)$. Formally,

$$
\begin{array}{lcl}
\lfloor \mathsf{tag}_d(u) \rfloor_1 & = & \mathsf{tag}_{d'}(\lfloor u \rfloor_1) \\
\lfloor \mathsf{untag}_d(u) \rfloor_1 & = & \mathsf{untag}_{d'}(\lfloor u \rfloor_1) \\
\lfloor \mathsf{senc}(u_1, u_2) \rfloor_1 & = & \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1), \lfloor u_2 \rfloor_1) \text{ if} \\
 & & \lfloor u_1 \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1)) \\
 & = & \mathsf{senc}_a(\lfloor u_1 \rfloor_1, \lfloor u_2 \rfloor_1) \text{ otherwise} \\
\lfloor \mathsf{sdec}(u_1, u_2) \rfloor_1 & = & \mathsf{sdec}_a(\lfloor u_1 \rfloor_1, \lfloor u_2 \rfloor_1) \\
\lfloor \mathsf{h}(u) \rfloor_1 & = & \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u \rfloor_1)) \text{ if} \\
 & & \lfloor u \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u \rfloor_1)) \\
 & = & \mathsf{h}_a(\lfloor u \rfloor_1) \text{ otherwise} \\
\lfloor u \rfloor_1 & = & u \text{ for a name or a variable.}
\end{array}
$$

We show the following.

(i) $\lfloor u \rfloor = \lfloor u \rfloor_1$ and $\lfloor v \rfloor = \lfloor v \rfloor_1$

(ii) $\lfloor u \rfloor$, $\lfloor v \rfloor$ are in head normal form

(iii) $u \neq v$ implies $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$

by induction on $\mathsf{max}(|u|, |v|)$. We can assume without loss of generality that $|v| \leq |u|$ and hence need only show items (i) and (ii) hold for the term u. For the base case, when u is a name or a variable, we have $u = \lfloor u \rfloor = \lfloor u \rfloor_1$ from which items (i) and (ii) are obvious. We also have $v = \lfloor v \rfloor$ and clearly $\lfloor u \rfloor = u \neq_{E^0} v = \lfloor v \rfloor$. For the induction step, we proceed by a case analysis on u.

case 1: $u = \mathsf{tag}_d(u_1)$.

(i) We have $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor u \rfloor_1 = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor_1)$. By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor u_1 \rfloor_1$ and thus $\lfloor u \rfloor =_{E^0} \lfloor u \rfloor_1$.

(ii) By definition $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and clearly $\lfloor u \rfloor$ is in head normal form.

(iii) We again do a case analysis on v. Assume $\lfloor u \rfloor =_E \lfloor u \rfloor$. By items (i) and (ii), the only interesting cases are when $v = \mathsf{tag}_d(v_1)$ or $v = \mathsf{sdec}(v_1, v_2)$. When $v = \mathsf{tag}_d(v_1)$, it must be the case that $u_1 \neq v_1$ because $u \neq v$. We have $\lfloor u \rfloor = \mathsf{tag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor v \rfloor = \mathsf{tag}_{d'}(\lfloor v_1 \rfloor)$. By the I.H. $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ and it follows that $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$.

When $v = \mathsf{sdec}(v_1, v_2)$, by item (i) we have $\lfloor v \rfloor = \lfloor v \rfloor_1$ and thus $\lfloor v \rfloor = \mathsf{sdec}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$. That is, $\lfloor u \rfloor \downarrow= \mathsf{tag}_{d'}(u'_1)$ and $\lfloor v \rfloor \downarrow= \mathsf{sdec}_a(v'_1, v'_2)$ and clearly $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$.

case 2: $u = \mathsf{untag}_d(u_1)$.

(i) We have $\lfloor u \rfloor = \mathsf{untag}_{d'}(\lfloor u_1 \rfloor)$ and $\lfloor u \rfloor_1 = \mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1)$. By the I.H. $\lfloor u_1 \rfloor =_{E^0} \lfloor u_1 \rfloor_1$ and thus $\lfloor u \rfloor =_{E^0} \lfloor u \rfloor_1$.

(ii) We again do a case analysis of u_1. First note that $u_1 \neq \mathsf{tag}_d(u_2)$ for any u_2. Otherwise we would have $u = \mathsf{untag}_d(\mathsf{tag}_d(u_2)) \to_E u_2$, contradicting the fact that u is in normal form. Now consider the case when $u_1 = \mathsf{sdec}(u_2, u_3)$. By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1 = \mathsf{sdec}_a(\lfloor u_2 \rfloor, \lfloor u_3 \rfloor)$. Then we have $\lfloor u \rfloor = \mathsf{untag}_{d'}(\mathsf{sdec}_a(\lfloor u_2 \rfloor, \lfloor u_3 \rfloor))$ which clearly does not rewrite in the root symbol. The remaining cases are straightforward.

(iii) By item (ii), $\lfloor u \rfloor = \mathsf{untag}_{d'}(\lfloor u_1 \rfloor)$ is in head normal form. We do a case analysis of v, with the only interesting case being when $v = \mathsf{untag}_d(v_1)$. Again by item (ii), we have $\lfloor v \rfloor = \mathsf{untag}_{d'}(\lfloor v_1 \rfloor)$ is in head normal form. It then suffices to show $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$. Because $u \neq v$, $u_1 \neq v_1$ and the I.H. yields $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$.

case 3: $u = \mathsf{senc}(u_1, u_2)$.

(i) By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1$ and $\lfloor u_2 \rfloor = \lfloor u_2 \rfloor_1$. From this we have

$$\lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor)) \text{ iff } \lfloor u_1 \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1))$$

and it follows that $\lfloor u \rfloor = \lfloor u \rfloor_1$.

(ii) By definition, $\lfloor u \rfloor = \mathsf{senc}_e(u_3', u_4')$ for some terms u_3', u_4' and $e \in \{a, b, c\}$. Clearly, $\mathsf{senc}_e(u_3', u_4')$ is in head normal form.

(iii) We do a case analysis on v. By item (ii), the only interesting case is when $v = \mathsf{senc}(v_1, v_2)$. In this case $u_1 \neq v_1$ or $u_2 \neq v_2$. By the I.H. $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ or $\lfloor u_2 \rfloor \neq_{E^0} \lfloor v_2 \rfloor$. Clearly if $\lfloor u \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1), \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$ (or vice-versa) then $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$. If $\lfloor u \rfloor = \mathsf{senc}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$ then the result follows from the fact that $\lfloor u \rfloor, \lfloor v \rfloor$ are in head normal form and $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ or $\lfloor u_2 \rfloor \neq_{E^0} \lfloor v_2 \rfloor$. When $\lfloor u \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor v \rfloor = \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor), \lfloor v_2 \rfloor)$ the result follows by a similar argument.

case 4: $u = \mathsf{sdec}(u_1, u_2)$.

(i) We do a case analysis on u_1. By the I.H. $\lfloor u_1 \rfloor$ is in head normal form and thus the only interesting case is when $u_1 = \mathsf{senc}(u_3, u_4)$. Assume for a contradiction that $\lfloor u_1 \rfloor =_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$. By definition, $\lfloor u_1 \rfloor = \mathsf{senc}_d(u_3', \lfloor u_4 \rfloor)$ for some term u_3'. In particular, this means that $\lfloor u_4 \rfloor =_{E^0} \lfloor u_2 \rfloor$. By the I.H. (contrapositive of item (iii)) $u_2 = u_4$. Then we have $u = \mathsf{sdec}(\mathsf{senc}(u_3, u_2), u_2) \to u_3$, which contradicts the fact that u is in normal form. That is, $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{senc}_d(\mathsf{sdec}_d(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor), \lfloor u_2 \rfloor)$ and $\lfloor u \rfloor = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$. By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1$ and $\lfloor u_2 \rfloor = \lfloor u_2 \rfloor_1$ from which it follows that $\lfloor u \rfloor = \lfloor u \rfloor_1$.

(ii) By item (i), $\lfloor u \rfloor = \lfloor u \rfloor_1$ and we have $\lfloor u \rfloor = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$. Notice that if $\lfloor u \rfloor$ is not in head normal form then $\lfloor u_1 \rfloor =_{E^0} \mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$ for some u_3. By the definition of $\lfloor \_ \rfloor$ and the fact that $\lfloor u_1 \rfloor$ is in head normal form (by the I.H.), it must be the case that $u_1 = \mathsf{senc}(u_3, u_4)$. Because u is in normal form, $u_1 \neq \mathsf{senc}(u_3, u_2)$ for any u_3. Otherwise we

would have $u = \mathsf{sdec}(\mathsf{senc}(u_3, u_2), u_2) \rightarrow_E u_3$. That is, $\lfloor u_1 \rfloor \neq_{E^0} \mathsf{senc}_a(\lfloor u_3 \rfloor, \lfloor u_2 \rfloor)$ for any u_3 and the result follows.

(iii) By items (i) and (ii), $\lfloor u \rfloor = \lfloor u \rfloor_1 = \mathsf{sdec}_a(\lfloor u_1 \rfloor, \lfloor u_2 \rfloor)$ is in head normal form. We do a case analysis on v, with the only interesting case being when $v = \mathsf{sdec}(v_1, v_2)$. Then again by items (i) and (ii), we have $\lfloor v \rfloor = \lfloor v \rfloor_1 = \mathsf{sdec}_a(\lfloor v_1 \rfloor, \lfloor v_2 \rfloor)$ is in head normal form. Because $u \neq v$ and either $u_1 \neq v_1$ or $u_2 \neq v_2$. By the I.H. either $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$ or $\lfloor u_2 \rfloor \neq_{E^0} \lfloor v_2 \rfloor$ from which the result follows.

case 5: $u = \mathsf{h}(u_1)$.

(i) By the I.H. $\lfloor u_1 \rfloor = \lfloor u_1 \rfloor_1$. From this we have $\lfloor u_1 \rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ iff $\lfloor u_1 \rfloor_1 =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor_1))$ and it follows that $\lfloor u \rfloor = \lfloor u \rfloor_1$.

(ii) By definition, $\lfloor u \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ or $\lfloor u \rfloor = \mathsf{h}_d(\lfloor u_1 \rfloor)$. Clearly, both $\mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ and $\mathsf{h}_d(\lfloor u_1 \rfloor)$ are in head normal form.

(iii) We do a case analysis on v. By item (ii), the only interesting case is when $v = \mathsf{h}(v_1)$. Because $u \neq v$, it must be the case that $u_1 \neq v_1$ and by the I.H. we have $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$. Clearly if $\lfloor u \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ and $\lfloor v \rfloor = \mathsf{h}_a(\lfloor u_1 \rfloor)$ (or vice-versa) then $\lfloor u \rfloor \neq_{E^0} \lfloor v \rfloor$. If $\lfloor u \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor u_1 \rfloor))$ and $\lfloor v \rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor v_1 \rfloor))$ or $\lfloor u \rfloor = \mathsf{h}_a(\lfloor u_1 \rfloor)$ and $\lfloor v \rfloor = \mathsf{h}_a(\lfloor v_1 \rfloor)$ then the result follows from the fact that $\lfloor u_1 \rfloor \neq_{E^0} \lfloor v_1 \rfloor$.

Lemma 5.21 *Let $u, v \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X})$. We have $u =_E v$ iff $\lfloor u \rfloor =_{E^0} \lfloor v \rfloor$.*

Proof. The proof follows from Lemma 5.19 and Lemma 5.20.

Let φ be a frame and σ be a substitution over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$. Define $\lfloor \varphi \rfloor = \{\lfloor w\varphi \rfloor \mid w \in \mathsf{dom}(\varphi)\}$ and $\lfloor \sigma \rfloor = \{\lfloor x\sigma \rfloor | x \in \mathsf{dom}(\sigma)\}$. Further define a function $\Diamond_\varphi : \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}_w) \rightarrow \mathcal{T}(\mathcal{F}_0, \mathcal{X}_w)$ as follows. This function is an adaptation of the one from [54].

$$
\begin{array}{lcl}
\Diamond_\varphi(\mathsf{tag}_d(r)) & = & \mathsf{tag}_{d'}(\Diamond_\varphi(r)) \\
\Diamond_\varphi(\mathsf{untag}_d(r)) & = & \mathsf{untag}_{d'}(\Diamond_\varphi(r)) \\
\Diamond_\varphi(\mathsf{senc}(r_1, r_2)) & = & \mathsf{senc}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)), \Diamond_\varphi(r_2)) \text{ if} \\
 & & \Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \\
 & & \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))\lfloor\varphi\rfloor \\
 & = & \mathsf{senc}_a(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2)) \text{ otherwise} \\
\Diamond_\varphi(\mathsf{sdec}(r_1, r_2)) & = & \mathsf{tag}_{d'}(\mathsf{sdec}_d(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2))) \text{ if} \\
 & & \Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \\
 & & \mathsf{senc}_d(\mathsf{sdec}(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2)), \Diamond_\varphi(r_2))\lfloor\varphi\rfloor \\
 & = & \mathsf{sdec}_a(\Diamond_\varphi(r_1), \Diamond_\varphi(r_2)) \\
\Diamond_\varphi(\mathsf{h}(r)) & = & \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r))) \text{ if} \\
 & & \Diamond_\varphi(r)\lfloor\varphi\rfloor =_{E^0} \\
 & & \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r))\lfloor\varphi\rfloor \\
 & = & \mathsf{h}_a(\Diamond_\varphi(r)) \text{ otherwise} \\
\Diamond_\varphi(w) & = & w \text{ for } w \in \mathsf{dom}(\varphi)
\end{array}
$$

We can lift $\Diamond_\varphi$ to actions in the following way. If $\alpha = (\tau, [\ell])$ then $\Diamond_\varphi(\alpha) = (\tau, [\ell])$. If $\alpha = (r, [\ell])$ then $\Diamond_\varphi(\alpha) = (\Diamond_\varphi(r), [\ell])$.

Lemma 5.22 *Let φ be a frame over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$ and $r \in \mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}}, \mathcal{X}_w)$. Then $\lfloor r\varphi\rfloor =_{E^0} \Diamond_\varphi(r)\lfloor\varphi\rfloor$.*

Proof. The proof is by induction on the structure of r. For the base case, when r is a frame variable w, we have $\Diamond_\varphi(w)\lfloor\varphi\rfloor = w\lfloor\varphi\rfloor = \lfloor w\varphi\rfloor$. For the induction step, we proceed by cases.

case 1: $r = \mathsf{tag}_d(r_1)$. We have $\lfloor\mathsf{tag}_d(r_1)\varphi\rfloor = \mathsf{tag}_{d'}(\lfloor r_1\varphi\rfloor)$ and $\Diamond_\varphi(\mathsf{tag}_d(r_1))\lfloor\varphi\rfloor = \mathsf{tag}_{d'}(\Diamond_\varphi(r_1)\lfloor\varphi\rfloor)$. By the I.H. $\lfloor r_1\varphi\rfloor =_{E^0} \Diamond_\varphi(r_1)\lfloor\varphi\rfloor$ and the case follows.

case 2: $r = \mathsf{h}(r_1)$. If $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))$ then

$$\Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))\lfloor\varphi\rfloor.$$

By the I.H. $\Diamond_\varphi(r_1)\lfloor\varphi\rfloor =_{E^0} \lfloor r_1\varphi\rfloor$ and we have $\lfloor r_1\varphi\rfloor =_{E^0} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor r_1\varphi\rfloor))$. By definition, this means that $\lfloor\mathsf{h}(r_1)\varphi\rfloor = \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor r_1\varphi\rfloor))$ and we have the following.

$$
\begin{array}{lcl}
\lfloor\mathsf{h}(r_1)\varphi\rfloor & = & \mathsf{h}_d(\mathsf{untag}_{d'}(\lfloor r_1\varphi\rfloor)) \\
 & =_{E^0} & \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)\lfloor\varphi\rfloor)) \\
 & = & \Diamond_\varphi(\mathsf{h}(r_1))\lfloor\varphi\rfloor
\end{array}
$$

If $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_a(\Diamond_\varphi(r_1))$ then we can show by a similar argument that $\lfloor \mathsf{h}(r_1)\varphi \rfloor = \mathsf{h}_a(\lfloor r_1\varphi \rfloor)$ and the case again follows by the I.H.

The remaining cases are similar.

Lemma 5.23 *Let φ_1, φ_2 be frames over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$. If $\lfloor \varphi_1 \rfloor \equiv_{E^0} \lfloor \varphi_2 \rfloor$ then $\varphi_1 \equiv_E \varphi_2$.*

Proof. We show the contrapositive. Assume $\varphi_1 \not\equiv_E \varphi_2$. By definition, there exist recipes r and r' such that $r\varphi_1 \neq_E r'\varphi_1$ and $r\varphi_2 =_E r'\varphi_2$ (or vice-versa). We show

$$\Diamond_{\varphi_1}(r)\lfloor \varphi_1 \rfloor \neq_E \Diamond_{\varphi_1}(r')\lfloor \varphi_1 \rfloor$$

and

$$\Diamond_{\varphi_1}(r)\lfloor \varphi_2 \rfloor =_E \Diamond_{\varphi_1}(r')\lfloor \varphi_2 \rfloor.$$

By Lemma 5.21, if $r\varphi_1 \neq_E r'\varphi_1$ then $\lfloor r\varphi_1 \rfloor \neq_{E^0} \rfloor r'\varphi_1 \rfloor$. Using Lemma 5.22, we have $\Diamond_\varphi(r)\lfloor \varphi_1 \rfloor \neq_{E^0} \Diamond_\varphi(r')\lfloor \varphi_1 \rfloor$. By a similar argument, we can derive $\Diamond_{\varphi_1}(r)\lfloor \varphi_2 \rfloor =_E \Diamond_{\varphi_1}(r')\lfloor \varphi_2 \rfloor$ and hence $\lfloor \varphi_1 \rfloor \not\equiv_{E^0} \lfloor \varphi_2 \rfloor$.

Lemma 5.24 *Let φ, φ' be frames over $\mathcal{T}(\mathcal{F}_{\mathsf{senc}} \cup \mathcal{F}_{\mathsf{tag}})$ such that $\lfloor \varphi \rfloor \equiv_{E^0} \lfloor \varphi' \rfloor$. Then $\Diamond_\varphi(r) = \Diamond_{\varphi'}(r)$.*

Proof. The proof is by induction on the structure of r. For the base case, when r is an frame variable w, we have $\Diamond_\varphi(w) = \Diamond_{\varphi'}(w) = w$. For the induction step, we proceed by cases.

case 1: $r = \mathsf{tag}_d(r_1)$. By definition, $\Diamond_\varphi(r) = \mathsf{tag}_{d'}(\Diamond_\varphi(r_1))$ and $\Diamond_{\varphi'}(r) = \mathsf{tag}_{d'}(\Diamond_{\varphi'}(r_1))$. From the I.H. $\Diamond_\varphi(r_1) = \Diamond_{\varphi'}(r_1)$ and it follows that $\Diamond_\varphi(r) = \Diamond_{\varphi'}(r)$.

case 2: $r = \mathsf{h}(r_1)$. By the I.H. we can define $r'_1 = \Diamond_\varphi(r_1) = \Diamond_{\varphi'}(r_1)$ and

$$r'_2 = \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1))) = \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\Diamond_{\varphi'}(r_1))).$$

Because $\lfloor \varphi \rfloor \equiv_{E^0} \lfloor \varphi' \rfloor$, we have $r'_1\lfloor \varphi \rfloor =_{E^0} r'_2\lfloor \varphi \rfloor$ iff $r'_1\lfloor \varphi' \rfloor =_{E^0} r'_2\lfloor \varphi' \rfloor$. This means that if $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_\varphi(r_1)))$ then $\Diamond_{\varphi'}(\mathsf{h}(r_1)) = \mathsf{h}_d(\mathsf{untag}_{d'}(\Diamond_{\varphi'}(r_1)))$ and if $\Diamond_\varphi(\mathsf{h}(r_1)) = \mathsf{h}_a(\Diamond_\varphi(r_1))$ then $\Diamond_{\varphi'}(\mathsf{h}(r_1)) = \mathsf{h}_a(\Diamond_{\varphi'}(r_1))$. In either case, the result follows by the I.H.

The remaining cases are similar.

Proposition 5.3 *Let $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$ and $u_1 \in \mathsf{st}(u)$. If $\mathsf{tests}^d(\mathcal{H}(u)\sigma)$ passes then $\mathsf{tests}^d(\mathcal{H}(u_1)\sigma)$ passes.*

Lemma 5.25 *Let $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$ and σ be a substitution over $\mathcal{F}^d_{\mathsf{senc}}$. If $\mathsf{tests}^d(\mathcal{H}(u)\sigma)$ passes then $\lfloor \mathcal{H}(u)\sigma \rfloor =_{E^0} u\lfloor \sigma \rfloor$.*

Proof. The proof is by induction on the structure of u. For the base case, when u is a name or variable we have $\lfloor \mathcal{H}(u)\sigma \rfloor = \lfloor u\sigma \rfloor = u\lfloor \sigma \rfloor$. For the induction step, we do a case analysis. By Proposition 5.3, we can apply the I.H. on any subterm of u.

case 1: $u = \mathsf{h}_d(u_1)$. By definition, $\mathcal{H}(u) = \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u_1)))$ and

$$\lfloor \mathcal{H}(u)\sigma \rfloor = \lfloor \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma)) \rfloor.$$

Let $v = \mathsf{tag}_d(\mathcal{H}(u_1)\sigma)$. We have

$$\begin{array}{rcl} \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v \rfloor)) & = & \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor \mathsf{tag}_d(\mathcal{H}(u_1)\sigma) \rfloor)) \\ & = & \mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor))) \\ & =_{E^0} & \mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor) \\ & = & \lfloor v \rfloor. \end{array}$$

The result is a consequence of the following derivation.

$$\begin{array}{rcll} \lfloor \mathcal{H}(u)\sigma \rfloor & = & \lfloor \mathsf{h}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma)) \rfloor & \\ & = & \mathsf{h}_d(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor))) & \\ & =_{E^0} & \mathsf{h}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor) & \\ & =_{E^0} & \mathsf{h}_d(u_1\lfloor \sigma \rfloor) & \text{(I.H.)} \\ & = & \mathsf{h}_d(u_1)\lfloor \sigma \rfloor & \end{array}$$

case 2: $u = \mathsf{senc}_d(u_1, u_2)$. By definition, $\mathcal{H}(u) = \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)), \mathcal{H}(u_2))$ and $\lfloor \mathcal{H}(u)\sigma \rfloor = \lfloor \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma), \mathcal{H}(u_2)\sigma) \rfloor$. Let $v = \mathsf{tag}_d(\mathcal{H}(u_1)\sigma)$. Identically to case 1, we can show that $\mathsf{tag}_{d'}(\mathsf{untag}_{d'}(\lfloor v \rfloor)) = \lfloor v \rfloor$. The result is a consequence of the following derivation.

$$\begin{array}{rcll} \lfloor \mathcal{H}(u)\sigma \rfloor & = & \lfloor \mathsf{senc}(\mathsf{tag}_d(\mathcal{H}(u_1)\sigma), \mathcal{H}(u_2)\sigma) \rfloor & \\ & = & \mathsf{senc}_d(\mathsf{untag}_{d'}(\lfloor \mathsf{tag}_d(\mathcal{H}(u_1)\sigma) \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor) & \\ & = & \mathsf{senc}_d(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\lfloor \mathcal{H}(u_1)\sigma \rfloor)), \lfloor \mathcal{H}(u_2)\sigma \rfloor) & \\ & =_{E^0} & \mathsf{senc}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor) & \\ & =_{E^0} & \mathsf{senc}_d(u_1\lfloor \sigma \rfloor, u_2\lfloor \sigma \rfloor) & \text{(I.H.)} \\ & = & \mathsf{senc}_d(u_1, u_2)\lfloor \sigma \rfloor & \end{array}$$

case 3: $u = \mathsf{sdec}_d(u_1, u_2)$. By definition $\mathcal{H}(u)\sigma = \mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma))$. Because $\mathsf{tests}^d(\mathcal{H}(u)\sigma)$ passes, it must be the case that

$$\mathsf{tag}_d(\mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma))) =_{E^0} \mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma).$$

The preceding equality only holds when

$$\mathcal{H}(u_1)\sigma =_{E^0} \mathsf{senc}(\mathsf{tag}_d(v), \mathcal{H}(u_2)\sigma) \tag{5.3}$$

for some term v. Therefore, by the definition of $\lfloor \_ \rfloor$, we have

$$\begin{array}{lll} \lfloor \mathcal{H}(u_1)\sigma \rfloor & =_{E^0} & \mathsf{senc}_d(\mathsf{untag}_{d'}(\mathsf{tag}_{d'}(v)), \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ & =_{E^0} & \mathsf{senc}_d(\lfloor v\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor) \end{array}$$

from which it follows that $\mathsf{senc}_d(\mathsf{sdec}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor)$

$$\begin{array}{ll} =_{E^0} & \mathsf{senc}_d(\mathsf{sdec}_d(\mathsf{senc}_d(v, \lfloor \mathcal{H}(u_2)\sigma \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor), \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ =_{E^0} & \mathsf{senc}_d(\lfloor v\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ =_{E^0} & \lfloor \mathcal{H}(u_1)\sigma \rfloor \end{array}$$

The result is consequence of the following.

$$\begin{array}{lll} \lfloor \mathcal{H}(u)\sigma \rfloor & = & \lfloor \mathsf{untag}_d(\mathsf{sdec}(\mathcal{H}(u_1)\sigma, \mathcal{H}(u_2)\sigma)) \rfloor \\ & = & \mathsf{untag}_{d'}(\mathsf{tag}_{d'}(\mathsf{sdec}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor))) \\ & =_{E^0} & \mathsf{sdec}_d(\lfloor \mathcal{H}(u_1)\sigma \rfloor, \lfloor \mathcal{H}(u_2)\sigma \rfloor) \\ & =_{E^0} & \mathsf{sdec}_d(u_1\lfloor \sigma \rfloor, u_2\lfloor \sigma \rfloor) \qquad \text{(I.H.)} \\ & = & \mathsf{sdec}_d(u_1, u_2)\lfloor \sigma \rfloor \end{array}$$

Define a function Θ on traces inductively as follows. If $|\overline{o}| = 0$ then $\Theta(\overline{o}) = \overline{o}$. Otherwise if $\overline{o} = \overline{o}_0 \xrightarrow{(\S,[\ell])} \mathsf{obs}(P, \varphi, \sigma)$ define

$$\Theta(\overline{o}) = \Theta(\overline{o}_0) \xrightarrow{(\tau,[\ell])} \mathsf{last}(\Theta(\overline{o}_0)) \xrightarrow{\Diamond_\varphi(\alpha)} \mathsf{obs}(P', \varphi', \sigma').$$

Let P be a process over $\mathcal{F}^b_{\mathsf{senc}} \cup \mathcal{F}^c_{\mathsf{senc}}$. Define a function $\Diamond_P : \mathsf{A} \to \mathsf{A}$ such that $\Diamond_P(\mathcal{A})(\overline{o}) = \mathcal{A}(\Theta(\overline{o}))$ if $\Theta(\overline{o}) \in \mathsf{Trace}([\![\lceil P \rceil^{\mathcal{A}}]\!])$. For substitutions σ, σ' and an equational theory E, we write $\sigma \cong_E \sigma'$ if $\mathsf{dom}(\sigma) = \mathsf{dom}(\sigma')$ and $x\sigma =_E x\sigma'$ for all $x \in \mathsf{dom}(\sigma)$.

Lemma 5.26 *Let P, Q be linear processes over $\mathcal{F}_{\mathsf{senc}}$ and W be an arbitrary interleaving of P^b and Q^c. If $\overline{o} =$*

$$\mathsf{obs}(\lceil W \rceil, \emptyset, \emptyset) \xrightarrow{(\tau,[\ell_1])} \mathsf{obs}(Q_1, \varphi_1, \sigma_1) \xrightarrow{\alpha_1} \ldots \xrightarrow{(\tau,[\ell_k])} \mathsf{obs}(Q_k, \varphi_k, \sigma_k) \xrightarrow{\alpha_k} \mathsf{obs}(\lceil W_k \rceil, \varphi_k, \sigma_k)$$

is a trace of $[\![\lceil W \rceil^{\mathcal{A}}]\!]$ then

$$\overline{o}_0 = \mathsf{obs}(W, \emptyset, \emptyset) \xrightarrow{\Diamond_{\varphi_1}(\alpha_1)} \ldots \xrightarrow{\Diamond_{\varphi_k}(\alpha'_k)} \mathsf{obs}(W'_k, \varphi'_k, \sigma'_k)$$

is a trace of $[\![W^{\Diamond_W(\mathcal{A})}]\!]$ such that $\varphi'_k \equiv_{E^0} \lfloor \varphi_k \rfloor$ and $\sigma'_k \equiv_{E^0} \lfloor \sigma_k \rfloor$.

Proof. The proof is by induction on k. The base case, $k = 0$, is trivial. For the induction step, let $\overline{o}_0 = \overline{o}_1 \xrightarrow{\alpha'_k} \mathsf{obs}(W', \varphi'_k, \sigma'_k)$ where $\mathsf{last}(\overline{o}_1) = \mathsf{obs}(W'', \varphi'_{k-1}, \sigma'_{k-1})$. By the I.H. $\overline{o}_1$ is a trace of $[\![W^{\Diamond_W(\mathcal{A})}]\!]$ such that $\varphi'_{k-1} \equiv_{E^0} \lfloor\varphi_{k-1}\rfloor$ and $\sigma'_{k-1} \equiv_{E^0} \lfloor\sigma_{k-1}\rfloor$. We proceed by cases.

case 1: α_k corresponds to an input action of the form $\mathtt{in}(x)$. By definition, α'_k also corresponds to an input action of the form $\mathtt{in}(x)$ and clearly $\overline{o}_0$ is a trace of $[\![W^{\Diamond(\mathcal{A})}]\!]$. We have $\varphi_k = \varphi_{k-1}$ and $\varphi'_k = \varphi'_{k-1}$ and therefore $\varphi'_k = \varphi'_{k-1} \equiv_{E^0} \lfloor\varphi_{k-1}\rfloor = \lfloor\varphi_k\rfloor$. By definition $\sigma_k = \sigma_{k-1} \cup \{x \mapsto r\varphi_{k-1}\}$ and $\sigma'_k = \sigma'_{k-1} \cup \{x \mapsto \Diamond_{\varphi_{k-1}}(r)\varphi'_{k-1}\}$. By the I.H. it suffices to show $x\sigma'_k =_{E^0} \lfloor x\sigma_k\rfloor$, which is a consequence of the derivation below.

$$\begin{array}{rcll} \lfloor x\sigma_k\rfloor & =_{E^0} & \lfloor r\varphi_{k-1}\rfloor & \text{(Lemma 5.21)} \\ & =_{E^0} & \Diamond_{\varphi_{k-1}}(r)\lfloor\varphi_{k-1}\rfloor & \text{(Lemma 5.22)} \\ & =_{E^0} & \Diamond_{\varphi_{k-1}}(r)\varphi'_{k-1} & \text{(I.H.)} \\ & = & x\sigma'_k & \end{array}$$

case 2: α_k corresponds to executing an assignment of the form $x := \mathcal{H}(u)$ for some $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$. By definition α'_k is also an assignment of the form $x := u$ and clearly $\overline{o}_0$ is a trace of $[\![W^{\Diamond(\mathcal{A})}]\!]$. Because $\overline{o}$ is a trace of $[\![\lceil W\rceil^{\mathcal{A}}]\!]$, it must be the case that $\mathsf{tests}^d(\mathcal{H}(u))$ passes. We also have $\varphi_k = \varphi_{k-1}$ and $\varphi'_k = \varphi'_{k-1}$ and therefore $\varphi'_k = \varphi'_{k-1} \equiv_{E^0} \lfloor\varphi_{k-1}\rfloor = \lfloor\varphi_k\rfloor$. By definition $\sigma_k = \sigma_{k-1} \cup \{x \mapsto \mathcal{H}(u)\sigma_{k-1}\}$ and $\sigma'_k = \sigma'_{k-1} \cup \{x \mapsto u\sigma'_{k-1}\}$. By the I.H. it suffices to show $x\sigma'_k =_{E_0} \lfloor x\sigma_k\rfloor$, which is a consequence of the derivation below.

$$\begin{array}{rcll} \lfloor x\sigma_k\rfloor & =_{E^0} & \lfloor\mathcal{H}(u)\sigma_{k-1}\rfloor & \text{(Lemma 5.21)} \\ & =_{E^0} & u\lfloor\sigma_{k-1}\rfloor & \text{(Lemma 5.25)} \\ & =_{E^0} & u\sigma'_{k-1} & \text{(I.H.)} \\ & =_{E^0} & x\sigma'_k & \end{array}$$

case 3: α_2 corresponds to executing a test of the form $[\mathcal{H}(u_1) = \mathcal{H}(u_2)]$ where $u_1, u_2 \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$. By definition α'_k is also a test of the form $[u_1 = u_2]$. Because $\overline{o}$ is a trace of $[\![\lceil W\rceil^{\mathcal{A}}]\!]$, it must be the case that $\mathsf{tests}^d(\mathcal{H}(u_1))$ and $\mathsf{tests}^d(\mathcal{H}(u_1))$ both pass. We also have $\varphi_k = \varphi_{k-1}$ and $\varphi'_k = \varphi'_{k-1}$ and therefore $\varphi'_k = \varphi'_{k-1} \equiv_{E^0} \lfloor\varphi_{k-1}\rfloor = \lfloor\varphi_k\rfloor$. We can similarly conclude that $\sigma'_k \equiv_{E^0} \lfloor\sigma_k\rfloor$. We know that $\mathcal{H}(u_1)\sigma_k =_E \mathcal{H}(u_2)\sigma_k$, from which we have the following derivation.

$$\begin{array}{rcll} \lfloor\mathcal{H}(u_1)\sigma_k\rfloor & =_{E^0} & \lfloor\mathcal{H}(u_2)\sigma_k\rfloor & \text{(Lemma 5.21)} \\ u_1\lfloor\sigma_k\rfloor & =_{E^0} & u_2\lfloor\sigma_k\rfloor & \text{(Lemma 5.25)} \\ u_1\sigma'_k & =_{E^0} & u_2\sigma'_k & \end{array}$$

That is, $\overline{o}_0$ is a trace of $[\![W^{\Diamond_W(\mathcal{A})}]\!]$

case 4: α_k is an output of the form $\mathtt{out}(\mathcal{H}(u))$ where $u \in \mathcal{T}(\mathcal{F}^d_{\mathsf{senc}}, \mathcal{X})$. By definition α'_k is also an output of the form $\mathtt{out}(u)$. Because $\overline{o}$ is a trace of $[\![\lceil W \rceil^{\mathcal{A}}]\!]$, it must be the case that $\mathsf{tests}^d(\mathcal{H}(u))$ passes. We also have $\sigma_k = \sigma_{k-1}$ and $\sigma'_k = \sigma'_{k-1}$ and therefore $\sigma'_k = \sigma'_{k-1} \equiv_{E^0} \lfloor \sigma_{k-1} \rfloor = \lfloor \sigma_k \rfloor$. By definition $\varphi_k = \varphi_{k-1} \cup \{w \mapsto \mathcal{H}(u)\sigma_k\}$ and $\varphi'_k = \varphi'_{k-1} \cup \{w \mapsto u\sigma'_k\}$. By the I.H. it suffices to show $w\varphi'_k =_{E_0} \lfloor w\varphi_k \rfloor$, which is a consequence of the derivation below.

$$\begin{array}{lll} \lfloor w\varphi_k \rfloor & =_{E^0} & \lfloor \mathcal{H}(u)\sigma_k \rfloor \quad \text{(Lemma 5.21)} \\ & =_{E^0} & u\lfloor \sigma_k \rfloor \quad \text{(Lemma 5.25)} \\ & =_{E^0} & u\sigma'_k \\ & = & w\varphi'_k \end{array}$$

case 5: α_2 corresponds to a new name creation of the form νn. This case is straightforward from the fact that α'_k is also a new name creation of the form νn and $\lfloor n \rfloor = n$.

Let $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot (D_1[\Box_1] \mid \ldots \mid D_n[\Box_n])$ (resp. $C'[\Box_1, \ldots, \Box_n] = \nu k'_1 \cdot \ldots \cdot \nu k'_m \cdot (D'_1[\Box_1] \mid \ldots \mid D'_n[\Box_n])$) be a context over $\mathcal{F}_{\mathsf{senc}}$ with labels from $\mathcal{L}_c$. Further let $B_1, \ldots, B_n$ (resp. $B'_1, \ldots, B'_n$) be basic processes over $\mathcal{F}_{\mathsf{senc}}$ with labels from $\mathcal{L}_b$. We will write ‡ when the conditions of Theorem 5.2 hold for the preceding processes. The following is a consequence of Lemmas 5.23, 5.24 and 5.26.

Proposition 5.4 *Assuming* ‡*, then*

$$(\lceil C^c[B^b_1, \ldots, B^b_n] \rceil,\ \lceil (C')^c[(B'_1)^b, \ldots, (B'_n)^b] \rceil)$$

is transposable to

$$(C^c[B^b_1, \ldots, B^b_n],\ (C')^c[(B'_1)^b, \ldots, (B'_n)^b]).$$

5.7 MULTI-SESSION PROTOCOLS

In this section, we extend our composition result to protocols that can run multiple sessions. Our focus will be on protocols that have a single occurrence of the replication operator appearing in the context. This restriction simplifies the statement of the results and proofs. However, it is possible to extend our results to protocols with a more general framework for replication. Formally, a context with replication is over the following grammar.

$$C[\Box_1, \ldots, \Box_m] ::= a^{\ell_1}_1 \cdot \ldots \cdot a^{\ell_n}_n . !^{\ell}(D_1[\Box_1] \mid \ldots \mid D_m[\Box_m])$$

where $a \in \{\nu x, (x := u)\}$. The semantics of this new replication operator are given in Figure 5.3, where $i \in \mathbb{N}$ is used to denote the smallest previously unused index. We will write $P(i)$ to denote that process that results from renaming each occurrence of $x \in \mathsf{vars}(P)$ to x^i for $i \in \mathbb{N}$. When $P(i)$ is relabeled freshly as in Figure 5.3, the new labels must all belong to the same equivalence class (that contains only those labels).

Figure 5.3 *Replication semantics.*

$$\frac{P(i) \text{ is relabeled freshly}}{(!^{\ell} P, \varphi, \sigma) \xrightarrow{(\tau,\ell)} \delta_{(P(i)|!^{\ell}P,\varphi,\sigma)}} \text{ REP}$$

Our semantics imposes an explicit variable renaming with each application of a replication rule. The reason for this is best illustrated through an example. Consider the process $!\mathtt{in}(x) \cdot P$ and the execution

$$(!\mathtt{in}(x) \cdot P, \emptyset, \emptyset) \rightarrow^* (\mathtt{in}(x) \cdot P \mid !\mathtt{in}(x) \cdot P, \varphi, \{x \mapsto u\} \cup \sigma)$$

where variable renaming does not occur. This execution corresponds to the attacker replicating $!\mathtt{in}(x) \cdot P$, running one instance of $\mathtt{in}(x) \cdot P$ and then replicating $!\mathtt{in}(x) \cdot P$ again. Note that, because x is bound at the end of the above execution, the semantics of the input action cause the process to deadlock at $\mathtt{in}(x)$. In other words, an attacker can only effective run one copy of $!\mathtt{in}(x) \cdot P$ for any process of the form $!\mathtt{in}(x) \cdot P$.

Our composition result must prevent messages from one session of a process from being confused with messages from another session. We achieve this by introducing an occurrence of $\nu\lambda$ directly following the replication operator. This freshly generated "session tag" will then be used to augment tags occurring in the composed processes. Recall that for any POMDPs $\mathcal{M}_1$ and $\mathcal{M}_2$, if $\mathcal{M}_1 \not\approx \mathcal{M}_2$ there exists an adversary $\mathcal{A}$ and trace $\overline{o}$ such that $\mathsf{prob}_{\mathcal{M}_1}(\overline{o}, \mathcal{A}) = \mathsf{prob}_{\mathcal{M}_2}(\overline{o}, \mathcal{A})$. This trace $\overline{o}$ must have finite length and subsequently $\mathcal{M}_1$ and $\mathcal{M}_2$ can only perform a bounded number of replication actions in $\overline{o}$. This means one can transform $\mathcal{A}, \overline{o}, \mathcal{M}_1, \mathcal{M}_2$ to an adversary $\mathcal{A}'$, trace $\overline{o}_0$ and POMDPs $\mathcal{M}_1'$ and $\mathcal{M}_2'$ such that $\mathsf{prob}_{\mathcal{M}_1'}(\overline{o}_0, \mathcal{A}') = \mathsf{prob}_{\mathcal{M}_2'}(\overline{o}_0, \mathcal{A}')$ where $\mathcal{M}_1'$ and $\mathcal{M}_2'$ do not contain replication. This is achieved by syntactically unrolling the replication operator $|\overline{o}|$ times in $\mathcal{M}_1$ (resp. $\mathcal{M}_2$). In the resulting process, every unrolling of $\mathcal{M}_1$ (resp. $\mathcal{M}_2$) generates a new parallel branch with fresh labels coming from a fresh equivalence class. The result below is a consequence of the preceding observation and Theorem 5.2. We again require the fixed equational theory $(\mathcal{F}_{\mathsf{senc}}, E_{\mathsf{senc}})$.

Theorem 5.3 *Let* $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot !\nu\lambda \cdot (D_1[\Box_1] \mid \ldots \mid D_n[\Box_n])$ *(resp.* $C'[\Box_1, \ldots, \Box_n] = \nu k'_1 \cdot \ldots \cdot \nu k'_m \cdot !\nu\lambda \cdot (D'_1[\Box_1] \mid \ldots \mid D'_n[\Box_n])$*) be a context over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_c$*. Further let* $B_1, \ldots, B_n$ *(resp.* $B'_1, \ldots, B'_n$*) be roles over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_b$*. For* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$ *and* $\sharp \notin \mathcal{F}_b \cup \mathcal{F}_c$*, assume that the following hold.*

1. $\mathsf{fv}(C) = \mathsf{fv}(C') = \emptyset$, $\mathsf{fv}(B_i) = \{x_i\}$ *and* $\mathsf{fv}(B'_i) = \{x'_i\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *and* $\mathsf{vars}(C') \cap \mathsf{vars}(B'_i) = \{x'_i\}$
3. $C[B_1, \ldots, B_n]$ *and* $C'[B'_1, \ldots, B'_n]$ *are ground*
4. $\lambda \notin \mathsf{vars}(C[B_1, \ldots, B_n]) \cup \mathsf{vars}(C'[B'_1, \ldots, B'_n])$
5. $C[B_1, \ldots, B_n] \models_{E_{\mathsf{senc}},1} \mathsf{secret}(x_1, \ldots, x_n)$ *and* $C'[B'_1, \ldots, B'_n] \models_{E_{\mathsf{senc}},1} \mathsf{secret}(x'_1, \ldots, x'_n)$
6. $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \approx C'[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}]$
7. $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot !(B_1 \mid \ldots \mid B_n) \approx \nu k \cdot (x'_1 := k) \cdot \ldots \cdot (x'_n := k) \cdot !(B'_1 \mid \ldots \mid B'_n)$

Then $\lceil \nu k_1 \cdot \ldots \cdot \nu k_m \cdot !\nu\lambda \cdot (D_1^{(c,\lambda)}[B_1^{(b,\lambda)}] \mid \ldots \mid D_n^{(c,\lambda)}[B_n^{(b,\lambda)}]) \rceil \approx \lceil \nu k'_1 \cdot \ldots \cdot \nu k'_m \cdot !\nu\lambda \cdot ((D'_1)^{(c,\lambda)}[(B'_1)^{(b,\lambda)}] \mid \ldots \mid (D'_n)^{(c,\lambda)}[(B'_n)^{(b,\lambda)}]) \rceil$.

CHAPTER 6: COMPOSITION OF SAFETY PROPERTIES

In this chapter, we extend the composition results from Chapter 5 to state-based safety properties. Our first composition result is for the composition of one session of protocol P and one session of protocol Q. We show that if P (in isolation) is secure with probability at least p (i.e., the shared secrets are not leaked) and Q is secure with probability at least q, then the composed protocol is secure with probability at least pq, provided the protocols are over disjoint equational theories (Theorem 6.1 on page 90). Although we exploit some techniques used in [54] to establish this result, there are important differences. First, the separation result in [54] does not carry over to the randomized setting. This is because an attack on the composition of P and Q is no longer a trace, but is instead a tree, as the protocol itself makes random choices. As a consequence, in different branches representing different resolutions of the randomized coin tosses, it is possible that the attacker may choose to send different messages (See example 6.1 on page 92). In such a case, an attack on the composition of P and Q cannot be separated into an attack on P and an attack on Q. Instead, we show that if there is an attack on the composition of P and Q then either we can extract an attack on P which succeeds with probability $> p$ or there is an attack on Q which succeeds with probability $> q$.

Our second composition result concerns multiple sessions of the composed protocol. Here, we would show that if n sessions of P are secure with probability at least p and n sessions of Q are secure with probability at least q then n sessions of the composed protocol are secure with probability at least pq, provided the protocol messages are tagged with the information of which protocol they belong to. Indeed, a similar result is claimed in [54] for the non-randomized protocols. Unfortunately, this result is not valid even for nonrandomized protocols and we exhibit a simple example which contradicts this desired result (See Example 6.3 on page 97). Essentially, the reason for this failure is that, in the claimed result, the n sessions of Q are assumed to generate fresh shared secrets in every session; but P may not be guaranteeing this freshness. Thus, messages of one session can get confused with messages of other sessions. We establish a weaker composition result in which we assume that the messages of each session of Q are tagged with a *unique* session identifier in addition to the protocol identifier. The use of session identifiers ensures that the messages of one session cannot be confused with other sessions.

Finally, we also consider the case for protocols containing an unbounded number of sessions. For this case, we observe that a composition result is only possible when P and Q are secure with probability exactly 1. This is because if m sessions of a protocol leak a

secret with probability $r > 0$ then by running mk sessions we can amplify the probability of leaking the secret. This probability approaches 1 as we increase k. We show that if an unbounded number of sessions of P are secure with probability 1 and an unbounded number of sessions of Q are secure with probability 1 then an unbounded number of sessions of the composed protocol are secure with probability 1, if the protocol messages are tagged with the information of which protocol they belong to and the messages of each session of Q are tagged with a *unique* session identifier.

6.1 COMPOSITION FRAMEWORK

Our composition setup for state-based safety properties will utilize the same machinery as Chapter 5. In particular, we will focus on the scenario in which a group of protocol participants, modeled by a context $C[\Box_0, \ldots, \Box_n]$, establishes a set of secrets to be used in a later phase of the protocol. The later phase will be modeled by a set of roles $B_1, \ldots, B_n$. As before, the context and roles will run over disjoint equational theories $(\mathcal{F}_c, E_c)$ and $(\mathcal{F}_b, E_b)$, respectively. Our main composition framework for safety properties follows. We will write $(\mathcal{F}, E)$ for $(\mathcal{F}_b \cup \mathcal{F}_c, E_b \cup E_c)$.

Theorem 6.1 *Let $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid \ldots \mid D_n[\Box_n])$ be a context over $\mathcal{F}_c$ with labels from $\mathcal{L}_c$ and $B_1, B_2, \ldots, B_n$ be roles over $\mathcal{F}_b$ with labels from $\mathcal{L}_b$. Further let $q_1, q_2 \in [0, 1]$, $x_s \in \bigcup_{i=1}^{n} vars(B_i) \setminus vars(C)$ and assume that the following hold.*

1. $\mathsf{fv}(C) = \emptyset$ *and* $\mathsf{fv}(B_i) \subseteq \{x_i\}$
2. $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) = \{x_i\}$ *for* $i \in \{1, \ldots, n\}$
3. $C[B_1, \ldots, B_n]$ *is ground*
4. $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \models_{E_c, q_1} \mathsf{secret}(x_1, \ldots, x_n)$ *where* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$
5. $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \models_{E_b, q_2} \mathsf{secret}(x_1, \ldots, x_n, x_s)$

Then $C[B_1, \ldots, B_n] \models_{E, q_1 q_2} \mathsf{secret}(x_s)$.

The proof of Theorem 6.1 will begin by transforming an attack on the composed protocol $C[B_1, \ldots, B_n]$ into an attack on on the protocol

$$C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \mid \nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n)$$

under a pure adversary (Definition 5.4). The techniques used in these adversary transformations are identical to the ones presented in Chapter 5. Using the pure adversary for the above protocol that breaks the security of the protocol with probability $> 1 - q_1 q_2$, we show that there exits an adversary the breaks the security of $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}]$ with probability $> 1 - q_1$ or an adversary the breaks the security of $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n)$ with probability $> 1 - q_2$. This result boils down to proving that an attacker for the asynchronous product of two POMDP's $\mathcal{M}_1$ and $\mathcal{M}_2$ can be transformed into an attacker for either $\mathcal{M}_1$ or $\mathcal{M}_2$. This is achieved by transforming the attacker $\mathcal{A}$ for $\mathcal{M}_1 \otimes \mathcal{M}_2$ into an attacker $\mathcal{A}'$ with the following two properties:

- $\mathcal{A}'$ executes all of the actions from $\mathcal{M}_1$ before executing any actions from $\mathcal{M}_2$.
- For any two executions $\rho, \rho' \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}'})$, if the projection of ρ and ρ' onto their components from $\mathcal{M}_i$ (for $i \in \{1, 2\}$) produces two equivalent traces and $\mathcal{A}'$ picks an action from $\mathcal{M}_i$, then $\mathcal{A}'(\mathsf{tr}(\rho)) = \mathcal{A}'(\mathsf{tr}(\rho'))$.
- $\mathcal{A}'$ derives a set secret values S with probability greater than or equal to $\mathcal{A}$.

The details of this construction are, which complete the proof of Theorem 6.1, are given in Section 6.2.

As a result of Theorem 6.1, one can reason about protocols composed sequentially by taking a context with a single basic context where a single hole appears at the end. The same is true for protocols composed in parallel, as given by Corollary 6.1. In this setting, one considers a context built over two basic contexts. One basic context contains only a hole, while the other contains no holes.

Corollary 6.1 *Let C be a role over $\mathcal{F}_c$ with labels from $\mathcal{L}_c$ and B be a role over $\mathcal{F}_b$ with labels from $\mathcal{L}_b$. Let $q_1, q_2 \in [0, 1]$ and assume that the following hold.*

1. $\mathsf{vars}(C) \cap \mathsf{vars}(B) = \emptyset$
2. $C \models_{E_c, q_1} \mathsf{secret}(x_c)$ *for* $x_c \in \mathsf{vars}(C)$
3. $B \models_{E_b, q_2} \mathsf{secret}(x_b)$ *for* $x_b \in \mathsf{vars}(B)$

Then $(C \mid B) \models_{E, q_1 q_2} \mathsf{secret}(x_b, x_c)$.

6.1.1 Difficulties with composition

In Section 5.1.4, we highlighted some of the major challenges in composing protocols with randomization. Because our composition framework for safety properties is the same as our composition framework for indistinguishability, many of the challenges from Section 5.1.4 extend naturally to the present setting. In the following examples, we present some additional difficulties that manifest themselves in the modular analysis of safety properties.

Example 6.1 *Consider the equational theories* $(\mathcal{F}_b, E_b)$ *and* $(\mathcal{F}_c, E_c)$ *where* $\mathcal{F}_b = \{\mathsf{h}/1\}$, $\mathcal{F}_c = \{\mathsf{c}/0\}$ *and* $E_b = E_c = \emptyset$. *Let* $C[\Box_1, \Box_2] = C_1 \cdot \Box_1 \mid C_2 \cdot \Box_2$ *be the context where* C_1 *and* C_2 *are as follows.*

$$\begin{aligned} C_1 &= \nu x_k \cdot (\mathtt{out}^1(x_k) +_{\frac{1}{2}} \mathtt{out}^2(\mathsf{c})) \\ C_2 &= \nu y_k \cdot (\mathtt{out}^3(y_k) +_{\frac{1}{2}} \mathtt{out}^4(\mathsf{c})) \end{aligned}$$

Essentially C_1 *generates* x_k *and with probability* $\frac{1}{2}$ *decides to reveal it.* C_2 *generates* y_k *and with probability* $\frac{1}{2}$ *decides to reveal it. In both cases, when the fresh values are not revealed, a constant is output in its place. Consider the roles* R_1 *and* R_2 *defined as follows.*

$$\begin{aligned} R_1 &= \mathtt{in}^5(x) \cdot [x = x_k] \cdot \nu x_s \cdot \mathtt{out}^6(x_s) \\ R_2 &= \mathtt{in}^7(y) \cdot [y = h(y_k)] \cdot \nu x_s \cdot \mathtt{out}^8(x_s) \end{aligned}$$

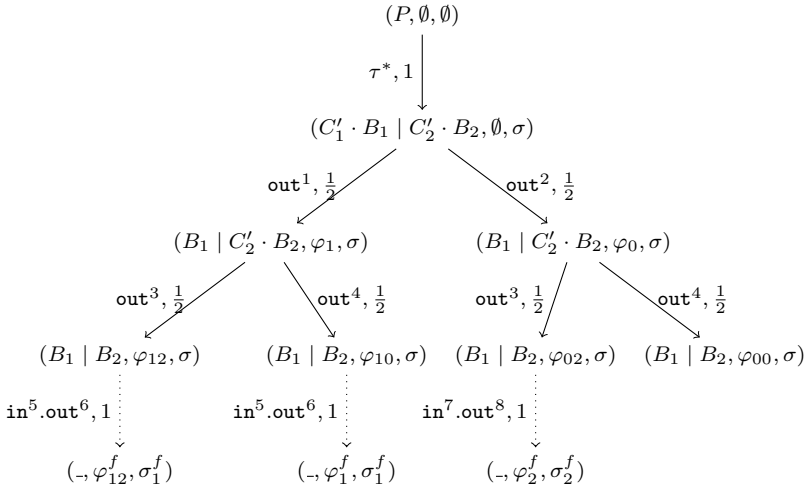
Let $P = C[B_1, B_2]$ *be a process and define the following.*

$$\begin{aligned} C_1' &= \mathtt{out}^1(x_k) +_{\frac{1}{2}} \mathtt{out}^2(\mathsf{c}) \\ C_2' &= \mathtt{out}^3(y_k) +_{\frac{1}{2}} \mathtt{out}^4(\mathsf{c}) \\ \sigma &= \{x_k \mapsto k_1, y_k \mapsto k_2\} \\ \varphi_0 &= \{w_1 \to \mathsf{c}\} \\ \varphi_1 &= \{w_1 \to k_1\} \\ \varphi_2 &= \{w_1 \to \mathsf{c}, w_2 \to k_2\} \\ \varphi_{00} &= \{w_1 \to \mathsf{c}, w_2 \to \mathsf{c}\} \\ \varphi_{10} &= \{w_1 \to k_1, w_2 \to \mathsf{c}\} \end{aligned}$$

$$\begin{aligned} \varphi_{12} &= \{w_1 \to k_1, w_2 \to k_2\} \\ \varphi_{02} &= \{w_1 \to k_1, w_2 \to \mathsf{c}\} \\ \sigma_1^f &= \{x_k \mapsto k_1, y_k \mapsto k_2, x \mapsto k_1, x_s \mapsto k_3\} \\ \sigma_2^f &= \{x_k \mapsto k_1, y_k \mapsto k_2, y \mapsto \mathsf{h}(k_1), x_s \mapsto k_3\} \\ \varphi_1^f &= \{w_1 \to k_1, w_2 \to \mathsf{c}, w_4 \mapsto k_3\} \\ \varphi_2^f &= \{w_1 \to \mathsf{c}, w_2 \to k_2, w_6 \mapsto k_3\} \\ \varphi_{12}^f &= \{w_1 \to k_1, w_2 \to k_2, w_4 \mapsto k_3\} \end{aligned}$$

The execution of P *shown in Figure 6.1 reveals* x_s *with probability* $\frac{3}{4}$. *Observe that the transitions out of the states labeling* $(B_1 \mid B_2, \varphi_1, \sigma)$ *involve transitions of* B_1 *while the transitions out of* $(B_1 \mid B_2, \varphi_2, \sigma)$ *involve transitions of* B_2. *If we try to fire the same transitions out of* $(B_1 \mid B_2, \varphi_2, \sigma)$ *as in* $(B_1 \mid B_2, \varphi_1, \sigma)$ *the process will deadlock because the attacker cannot deduce* x_k *in* φ_2. *From this, it is easy to see that the execution shown in Figure 6.1 cannot be written as an interleaving of one execution of* $C_1 \mid C_2$ *and one*

Figure 6.1 *Execution of P.* The solid edges are transitions of the context C and dotted edges are transitions of the basic processes B_1, B_2. For convenience, the edges in the drawn execution tree may compose of more than 1 action. The recipes used in in^3 and in^5 are w_1 and $h(w_2)$ respectively. The transition probabilities also label the edges.



execution of $B_1 \mid B_2$. As a result, the proof technique of [54] is not immediately applicable. Nevertheless, we will be able to show that P keeps x_s secret with probability at least $\frac{1}{4}$.

Notice that the execution of P shown in Figure 6.1, the attacker performs different actions depending of the result of coin toss made by C_1. When C_1 outputs a nonce, B_1 is scheduled before B_2. When C_1 outputs the constant c, B_2 is executed first. Such an attack is valid, even when considering our restricted class of adversaries. The reason is that the attacker can infer the result of the coin toss in C_1 by observing what is output.

It is important to point out that the security guarantees of the composed process may in fact be stronger than what we can prove utilizing Theorem 6.1. This is because we always assume the worst case in that context assigns the same secret values to each basic process. As a result, our composition result will in some cases lead to only an under-approximation on the probability that a set of variables is kept secret, as shown by the following example.

Example 6.2 *Consider the signatures $\mathcal{F}_b = \{\mathsf{h}/1\}$ and $\mathcal{F}_c = \{\}$ with empty equational theories and the context defined below.*

$$C[\Box_1, \Box_2] = \nu k_1 \cdot \nu k_2 \cdot (([x_1 := k_1] +_{\frac{1}{2}} [x_1 := k_2]) \cdot \Box_1 \mid [x_2 := k_2] \cdot \Box_2)$$

Essentially, the context generates shared secrets x_1 and x_2 for two sub-protocols $\Box_1$ and $\Box_2$ to be run in parallel. For the sub-protocol $\Box_1$, it sets the secret x_1 to k_1 with probability $\frac{1}{2}$ and to k_2 with probability $\frac{1}{2}$. In the second sub-protocol, the shared secret x_2 is set to k_2.

Now consider the sub-protocols B_1 and B_2 defined as follows.

$$\begin{aligned} B_1 &= \mathtt{out}(\mathsf{h}(x_1)) +_{\frac{1}{2}} 0 \\ B_2 &= \mathtt{in}(z) \cdot [z = \mathsf{h}(x_2)] \cdot \nu x_s \cdot \mathtt{out}(x_s) \end{aligned}$$

B_1 outputs $\mathsf{h}(x_1)$ with probability $\frac{1}{2}$ and with probability $\frac{1}{2}$ does nothing. B_2 checks if the attacker can construct $\mathsf{h}(x_2)$ before revealing x_s. It is easy to see that $C[B_1, B_2]$ reveals x_s with probability $\frac{1}{4}$. This is because the attacker can construct $\mathsf{h}(x_2)$ when x_1 and x_2 are equal (which happens with probability $\frac{1}{2}$) and when B_1 reveals $\mathsf{h}(x_1)$ (which also happens with probability $\frac{1}{2}$). Infact, we can easily show that $C[B_1, B_2]$ keeps x_s secret with probability exactly $\frac{3}{4}$. However, Theorem 6.1 can only show $C[B_1, B_2]$ keeps x_s secret with probability $\frac{1}{2}$, since in our composition results, we assume that x_1 and x_2 get the same secret name.

6.2 SAFETY PROPERTIES IN PRODUCT POMDPS

We first fix some notation that will be used throughout the remainder of this section. Let $\mathcal{M}_i = (Z_i, z_i^s, \mathsf{Act}_i, \Delta_i, \equiv_i)$ be a POMDP for $i \in \{1, 2\}$. We will assume that $Z_1 \cap Z_2 = \emptyset$ and $\mathsf{Act}_1 \cap \mathsf{Act}_2 = \emptyset$. We will assume that $\mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ is finite and every execution is of finite length. The projection function π_i on states, observation, executions and traces will be the same as in Section 5.3.

Let $S_1 \subseteq Z_1$, $S_2 \subseteq Z_2$ and $S = S_1 \cup S_2$. We write $\mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(S, \mathcal{A})$ to denote the maximal probability p such that $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}} \models_p S$. Intuitively, S can be thought of the attack states (states in which the attacker can derive some secret value) in $\mathcal{M}_1 \otimes \mathcal{M}_2$ and $\mathsf{prob}_{M_1 \otimes M_2}(S, \mathcal{A})$ is the exact probability of reaching an attack state under the attacker $\mathcal{A}$. An execution $\rho \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ is said to be distinguishing if there exist one step extensions ρ_1 and ρ_2 of ρ such that $\mathsf{last}(\rho_1) \not\equiv \mathsf{last}(\rho_2)$. An $L \in \mathbb{N}$ is said to be a distinguishing level in $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$ if $L = \min(|\rho_1|, \ldots, |\rho_n|)$ for all distinguishing executions $\rho_1, \ldots, \rho_n \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$. When $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$ contains no distinguishing executions, the distinguishing level is ∞. For ease of notation, we will write $\bar{i}$ to denote the only element of $\{1, 2\} \setminus \{i\}$.

Definition 6.1 *An attacker $\mathcal{A}$ for $\mathcal{M}_1 \otimes \mathcal{M}_2$ is said to be process determined if, for any $\rho, \rho' \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$, if $\mathsf{tr}(\pi_i(\rho)) = \mathsf{tr}(\pi_i(\rho'))$ then $\mathcal{A}(\mathsf{tr}(\rho)) = \mathcal{A}(\mathsf{tr}(\rho'))$.*

Definition 6.2 *An execution $\rho \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ is said to be $(\mathcal{M}_i, \mathcal{M}_{\bar{i}})$-sequential if there exists a $k \in \mathbb{N}$ such that*

$$\rho = z_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_k} z_k \xrightarrow{\alpha_{k+1}} \ldots \xrightarrow{\alpha_{k+m}} z_{k+m}$$

where $\alpha_1, \ldots, \alpha_k \in \mathsf{Act}_i$ *and* $\alpha_{k+1}, \ldots, \alpha_{k+m} \in \mathsf{Act}_{\bar{i}}$. *An attacker* $\mathcal{A}$ *is called* $(\mathcal{M}_i, \mathcal{M}_{\bar{i}})$*-sequential if any execution in* $\mathsf{Exec}((\mathcal{M}_i \otimes \mathcal{M}_{\bar{i}})^{\mathcal{A}})$ *is* $(\mathcal{M}_i, \mathcal{M}_{\bar{i}})$*-sequential.*

Lemma 6.1 *For any* $(\mathcal{M}_i, \mathcal{M}_{\bar{i}})$*-sequential attacker* $\mathcal{A}$ *of* $\mathcal{M}_1 \otimes \mathcal{M}_2$*, there exists a sequential and process determined attacker* $\mathcal{A}'$ *such that* $\mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}) \leq \mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}')$.

Proof. Let $\rho = z_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_k} z_k$ be an execution in $\mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ such that $\alpha_1, \ldots, \alpha_k \in \mathsf{Act}_1$ and for any one step extension $\rho' = \rho \xrightarrow{\alpha} z$, $\alpha \in \mathsf{Act}_2$. We know that for any pair of one step extensions ρ_1 and ρ_2 of ρ, $\mathsf{tr}(\pi_2(\rho_1)) = \mathsf{tr}(\pi_2(\rho_2))$. That is, we can define an attacker $\mathcal{A}'$ on $\mathcal{M}_2$ with respect to the execution ρ by induction on the number of transitions in $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$ starting from the prefix ρ. For the base case, let $z_2 \in \mathsf{Exec}((\pi_2(\rho))^{\mathcal{A}'})$ where $\mathsf{last}(\rho) = (z_1, z_2)$. For the inductive step, let $\rho \xrightarrow{\alpha} \rho' \xrightarrow{\alpha'} z$ be an execution of $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$. Inductively, we have an execution $\rho'' \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}'})$ for $\rho \xrightarrow{\alpha} \rho'$. Define $\mathcal{A}'(\mathsf{tr}(\rho'')) = \alpha'$.

Let Θ be the set of all executions in $\mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ having the property of execution ρ. For each $\rho_j \in \Theta$, let $\mathcal{A}_j$ be the scheduler for $\mathcal{M}_2$ defined with respect to ρ_i. Let $j \in \{1, \ldots, n\}$ and m be the index of some execution in Θ such that $\mathsf{prob}_{\mathcal{M}_2}(S, \mathcal{A}_m) \geq \mathsf{prob}_{\mathcal{M}_2}(S, \mathcal{A}_j)$ for all j. Define the attacker $\mathcal{A}''$ for $\mathcal{M}_1 \otimes \mathcal{M}_2$ that behaves like $\mathcal{A}$ until reaching an execution of Θ and then behaves like $\mathcal{A}_m$ on the remaining $\mathcal{M}_2$ component. Clearly, $\mathcal{A}''$ is both sequential and process determined. Furthermore, because $\mathcal{M}_2$ is executed with respect to the maximal adversary $\mathcal{A}_m$ of $\mathcal{M}_2$ for every execution in $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}''}$, we have $\mathsf{prob}_{(M_1 \otimes M_2)}(S, \mathcal{A}) \leq \mathsf{prob}_{(M_1 \otimes M_2)}(S, \mathcal{A}'')$.

Proposition 6.1 *Let* L *be the distinguishing level of* $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$ *and* $k \leq L$*. There exists an attacker* $\mathcal{A}'$ *for* $\mathcal{M}_1 \otimes \mathcal{M}_2$ *such that for any* $\rho \in \mathsf{Exec}(\mathcal{M}_1 \otimes \mathcal{M}_2)$ *where* $|\rho| = k$*,* ρ *is* $(\mathcal{M}_i, \mathcal{M}_{\bar{i}})$*-sequential and* $\mathsf{prob}_{\mathcal{M}_1 \otimes \mathcal{M}_2}(S, \mathcal{A}) = \mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}')$.

Lemma 6.2 *For any attacker* $\mathcal{A}$ *of* $\mathcal{M}_1 \otimes \mathcal{M}_2$*, there exists an* (M_1, M_2)*-sequential and processes determined attacker* $\mathcal{A}'$ *such that* $\mathsf{prob}_{(\mathcal{M}_1 \times \mathcal{M}_2)}(S, \mathcal{A}) \leq \mathsf{prob}_{(\mathcal{M}_1 \times \mathcal{M}_2)}(S, \mathcal{A}')$.

Proof. The proof is by induction on the number of distinguishing executions in $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$. The base case, when there are no distinguishing executions, follows by Proposition 6.1 and Lemma 6.1. For the inductive step, let the number of distinguishing executions in $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}}$ be $d + 1$ and let the distinguishing level be L. For any $\rho_1, \rho_2 \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}})$ where $|\rho_1| = |\rho_2| = L$, $\mathcal{A}(\mathsf{tr}(\rho_1)) = \mathcal{A}(\mathsf{tr}(\rho_2))$. Fix $\mathcal{A}(\mathsf{tr}(\rho_1)) = \alpha$. We proceed by cases.

Case 1: $\alpha \in \mathsf{Act}_1$. By Proposition 6.1, there exists an adversary $\mathcal{A}'$ such that for any $\rho \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}'})$ where $|\rho| = L$, ρ is $(\mathcal{M}_1, \mathcal{M}_2)$-sequential and $\mathsf{prob}_{(\mathcal{M}_1 \times \mathcal{M}_2)}(S, \mathcal{A}) = \mathsf{prob}_{(\mathcal{M}_1 \times \mathcal{M}_2)}(S, \mathcal{A}')$. Let the distinguishing level of this new DTMC $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}'}$ be L'

and let $\Theta = \{\rho \mid \rho \in \mathsf{Exec}((\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}'})$ and $|\rho| = L'\}$. Assume without loss of generality that $\Theta = \Theta_{e_1} \uplus \Theta_{e_1}$ where $\Theta_{e_i} = \{\rho \mid \rho \in \Theta$ and $\mathsf{last}(\rho) \in e_i\}$ for some e_i in $\equiv$. Let $f_i : \{1, \ldots, |\Theta_{e_i}|\} \mapsto \Theta_{e_1}$ be a bijection. For $j \in \{1, \ldots, |\Theta_{e_i}|\}$, let κ_j be the probability of event $f_i(j)$ in $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}'}$ and let κ_{e_i} be the sum of the probabilities of the events $f_i(1), \ldots, f_i(j)$. Define the POMDP $\mathcal{M}_{e_i} = (Z_i, z_i^s, \mathsf{Act}_i \cup \{\alpha_{new}\}, \Delta_i', \equiv_i)$, where $\alpha_{new} \notin \mathsf{Act}_1 \cup \mathsf{Act}_2$ and Δ_i' is the same as Δ_i with addition of $\Delta_i'(z_i^s, \alpha_{new}) = \mu$ where $\mu(\mathsf{last}(f_i(j))) = \kappa_j/\kappa_{e_i}$ for all j. We define an attacker $\mathcal{A}_{e_i}$ on $\mathcal{M}_{e_i} \otimes \mathcal{M}_2$ from $\mathcal{A}'$ such that

$$z_s \xrightarrow{\alpha_{new}} z_{L'} \xrightarrow{\alpha_{L'+L}} \ldots \xrightarrow{\alpha_{L'+m}} z_{L'+m} \tag{6.1}$$

is an execution of $(\mathcal{M}_{e_i} \otimes \mathcal{M}_2)^{\mathcal{A}_{e_i}}$ iff

$$z_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_{L'}} z_{L'} \xrightarrow{\alpha_{L'+L}} \ldots \xrightarrow{\alpha_{L'+m}} z_{L'+m} \tag{6.2}$$

is an execution of $(\mathcal{M}_i \otimes \mathcal{M}_2)^{\mathcal{A}_i'}$. Notice that the number of distinguishing states in $(\mathcal{M}_{e_i} \otimes \mathcal{M}_2)^{\mathcal{A}_{e_i}}$ is d. By our inductive assumption, there exists a process determined and $(\mathcal{M}_1, \mathcal{M}_2)$-sequential attacker $\mathcal{A}_{e_i}'$ for $\mathcal{M}_{e_i} \otimes \mathcal{M}_2$ where $\mathsf{prob}_{(\mathcal{M}_{e_i} \otimes \mathcal{M}_2)}(S, \mathcal{A}_{e_i}) \leq \mathsf{prob}_{(\mathcal{M}_{e_i} \otimes \mathcal{M}_2)}(S, \mathcal{A}_{e_i}')$. Using $\mathcal{A}_{e_1}'$ and $\mathcal{A}_{e_2}'$, we construct a scheduler $\mathcal{A}_e$ for $\mathcal{M}_1 \otimes \mathcal{M}_2$ inductively as follows. For any $\rho \in (\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}_e}$ such that $|\rho| < L'$, let $\mathcal{A}_e(\mathsf{tr}(\rho)) = \mathcal{A}'(\mathsf{tr}(\rho))$. For any $\rho \in (\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}_e}$ such that $|\rho| \geq L'$, $\mathcal{A}_e$ behaves like $\mathcal{A}_{e_i}'$ if the initial prefix of ρ is in Θ_{e_i}. Clearly, $\mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}') \leq \mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}_e)$. Observe that $\mathcal{A}_e$ is a sequential adversary for $\mathcal{M}_1 \otimes \mathcal{M}_2$. Because $\mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}) = \mathsf{prob}_{(\mathcal{M}_1', \mathcal{M}_2)}(S, \mathcal{A}') \leq \mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}_e)$ we can apply Lemma 6.1 to conclude that there exits a sequential and process determined adversary $\mathcal{A}_e'$ for $\mathcal{M}_1 \otimes \mathcal{M}_2$ such that $\mathsf{prob}_{(\mathcal{M}_1 \otimes \mathcal{M}_2)}(S, \mathcal{A}) \leq \mathsf{prob}_{(M_1 \otimes M_2)}(S, \mathcal{A}_e')$.

Case 2: $\alpha \in \mathsf{Act}_2$). Follows by a similar argument as case 1.

Proposition 6.2 *If $\mathcal{M}_1 \models_{q_1} S_1$ and $\mathcal{M}_2 \models_{q_2} S_2$ then for any $(\mathcal{M}_1, \mathcal{M}_2)$-sequential and process determined scheduler $\mathcal{A}$ for $\mathcal{M}_1 \otimes \mathcal{M}_2$, $(\mathcal{M}_1 \otimes \mathcal{M}_2)^{\mathcal{A}} \models_{q_1 q_2} S$.*

6.3 EXTENSIONS OF THE COMPOSITION FRAMEWORK

In this section, we expand on the composition result from Section 6.1. We begin by showing that the result continues to hold when the protocols share a common signature containing primitives of symmetric encryption/decryption and hashing. Our tagging scheme is identical to the one Chapter 5. The following is a consequence of Theorem 6.1 and Lemmas 5.23, 5.24 and 5.26.

Theorem 6.2 *Let* $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid \ldots \mid D_n[\Box_n])$ *be a context over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_c$ *and* $B_1, B_2, \ldots, B_n$ *be roles over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_b$*. Further let* $q_1, q_2 \in [0,1]$*,* $x_s \in \bigcup_{i=1}^{n} \mathsf{vars}(B_i) \setminus \mathsf{vars}(C)$ *and assume that the following hold.*

- $\mathsf{fv}(C) = \emptyset$ *and* $\mathsf{fv}(B_i) = \{x_i\}$
- $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, \ldots, n\}$
- $C[B_1, \ldots, B_n]$ *is ground*
- $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \models_{E_{\mathsf{senc}}, q_1} \mathsf{secret}(x_1, \ldots, x_n)$ *where* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$
- $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \models_{E_{\mathsf{senc}}, q_2} \mathsf{secret}(x_1, \ldots, x_n, x_s)$

Then $\lceil C^c[B_1^b, \ldots, B_n^b] \rceil \models_{E_{\mathsf{senc}} \cup E_{\mathsf{tag}}, q_1 q_2} \mathsf{secret}(x_s)$.

We can also extend our composition result to protocols that can run multiple sessions.[1] We will begin by considering processes that contain only a bounded version of the replication operator. The bounded replication operator has an explicit bound that limits the number of times a process can replicate. As in Section 5.7, we will only consider processes that a single occurrence of the replication operator.

In our first such result, we will show that if the protocols $C = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot !_n(C[\Box_1] \mid \ldots \mid C[\Box_\ell])$ and $!_n(B_1 \mid \ldots \mid B_\ell)$ are proven secure with probability at least p and q, respectively, then the composition $\nu k_1 \cdot \ldots \cdot \nu k_m \cdot !_n(C[B_1] \mid \ldots \mid C[B_\ell])$ is secure with probability at least pq. The result requires that protocol messages are tagged with both a protocol identifier and a unique session identifier. A similar result (with the absence of the session identifier), has been claimed in [54] for non-randomized protocols (with p and q both being 1). However, we discovered a simple counterexample, which works for the case of two sessions. Essentially the reason for this attack is that protocol messages from one session can be confused with messages from the other session.

Example 6.3 *Consider the equational theories* $(\mathcal{F}_b, E_b)$ *and* $(\mathcal{F}_c, E_c)$ *with signatures* $\mathcal{F}_b = \{\mathsf{h}/1, \mathsf{c}/0\}$ *and* $\mathcal{F}_a = \{\}$ *and equations* $E_a \cup E_b = \emptyset$*. We will consider two sessions of the composed protocol. Let* P *be the process below.*

$$\begin{aligned} P &= \nu k_1 \cdot \nu k_2 \cdot !_2(P_1 \mid P_2) \\ P_1 &= (x_k := k_1) \\ P_2 &= (y_k := k_2) \end{aligned}$$

[1] n sessions of P will be denoted by $!_n P$.

Let Q be the process below.

$$\begin{aligned} Q &= !_2(\nu k \cdot ((x_k := k) \cdot Q_1 \mid (y_k := k) \cdot Q_2)) \\ Q_1 &= \mathtt{in}(y) \cdot ([y = c] \cdot \mathtt{out}^l(h(x_k)) \mid [y = h(x_k)] \cdot \nu x_s \cdot \mathtt{out}^{l'}(x_s)) \\ Q_2 &= 0 \end{aligned}$$

Clearly, P keeps x_k and y_k secret with probability 1 and Q keeps x_k, y_k and x_s secret with probability 1. Theorem 3 from [54] would imply that x_s is kept secret by $W = \nu k_1 \cdot \nu k_2 \cdot !_2(P_1 \cdot Q_1 \mid P_2 \cdot Q_2)$ in both sessions of the protocol. However, we can show that this is not the case. The reason is as follows. In both sessions of the composed protocol, x_k gets the same value. In the first session of the composed protocol, when y is input by Q_1, attacker sends the constant c. Thereafter, the attacker learns $\mathsf{h}(x_k)$ because Q_1 outputs it. In the second session of the composed protocol, the attacker sends $\mathsf{h}(x_k)$ to Q_1; the check $[y = \mathsf{h}(x_k)]$ succeeds and the attacker learns x_s in this session.

In process calculus terms, this attack can be realized by the execution:

$$(W, \emptyset, \emptyset) \rightarrow^* (W', \emptyset, \sigma') \rightarrow^* (0, \varphi'', \sigma'')$$

where

$$\begin{aligned} W' &= (Q_1^1 \mid Q_1^2) \\ \sigma' &= \{k_1 \mapsto n_1, k_2 \mapsto n_2, x_k^1 \mapsto n_1, y_k^1 \mapsto n_2, x_k^2 \mapsto n_1, y_k^2 \mapsto n_2\} \\ \sigma'' &= \sigma' \cup \{y^1 \mapsto c, y^2 \mapsto \mathsf{h}(n_1), x_s^2 \mapsto n_3\} \\ \varphi'' &= \{w_l \mapsto \mathsf{h}(n_1), w_{l'} \mapsto n_3\} \end{aligned}$$

Note above that we have used superscripts on variables x_k, y_k, y and x_s in the substitutions to indicate their values in different sessions. We have also indexed frame variables by the label of the corresponding output action they were generated from. Essentially in this execution in $(W', \emptyset, \sigma')$, P is finished in both sessions and assigned x_k and y_k the same values in both sessions. The role Q_2 is also finished in both sessions. Q_1^1 is the first session of Q_1 and Q_1^2 is the second session of Q_1. Now in Q_1^1, the attacker inputs c for y resulting in Q_1^1 leaking $\mathsf{h}(n_1)$. In Q_1^2, the attacker can input $\mathsf{h}(n_1)$ and learn the value of x_s generated.

Formally, a context containing bounded replication is defined as

$$C[\Box_1, \ldots, \Box_m] := a_1^{\ell_1} \cdot \ldots \cdot a_n^{\ell_n} \cdot !_n^{\ell}(D_1[\Box_1] \mid^{\ell_{n+1}} D_2[\Box_2] \mid^{\ell_{n+2}} \ldots \mid^{\ell_{n+m-1}} D_m[\Box_m])$$

where $a \in \{\nu x, (x := u)\}$ and $n \geq 2$ is a natural number. The semantics for this bounded replication operator is given in Figure 6.2, where $i, j \in \mathbb{N}$ are used to denoted the smallest previously unused indices. We will use $P(i)$ to denote that process that results from renaming

each occurrence of $x \in \mathsf{vars}(P)$ to x^i for $i \in \mathbb{N}$. When $P(i)$ or $P(j)$ is relabeled freshly as in Figure 6.2, the new labels must all belong to the same equivalence class (that contains only those labels). The notation x^* denotes the infinite set $\{x^0, x^1, x^2, \ldots\}$.

Figure 6.2 *Bounded replication semantics.*

$$\frac{n > 2 \quad \ell' \text{ is a fresh label} \quad P(i) \text{ is relabeled freshly}}{(!^{\ell}_{n} P, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(P(i)|^{\ell'} !^{\ell}_{n-1} P, \varphi, \sigma)}} \text{ B-REP}$$

$$\frac{\ell' \text{ is a fresh label} \quad P(i), P(j) \text{ are relabeled freshly}}{(!^{\ell}_{2} P, \varphi, \sigma) \xrightarrow{(\tau,[\ell])} \delta_{(P(i)|^{\ell'} P(j), \varphi, \sigma)}} \text{ B-REP}_{n=2}$$

As demonstrated in Example 6.3, our composition result must prevent messages from one session of a process with bounded replication from being confused with messages from another sessions. We achieve this in the following way. Our composed processes will contain an occurrence of $\nu\lambda$ directly following the occurrence of a bounded replication operator. This freshly generated "session tag" will then be used to augment tags occurring in the composed processes. We have the following result.

Theorem 6.3 *Let* $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m \cdot !_w \nu\lambda \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid \ldots \mid D_n[\Box_n])$ *for* $w \in \mathbb{N}$ *be a context over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_c$ *and* $B_1, B_2, \ldots, B_n$ *be roles over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_b$ *Further let* $q_1, q_2 \in [0,1]$, $x_s \in \bigcup_{i=1}^{n} \mathsf{vars}(B_i) \setminus \mathsf{vars}(C)$ *and assume the following hold.*

- $\mathsf{fv}(C) = \emptyset$ *and* $\mathsf{fv}(B_i) \subseteq \{x_i\}$
- $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, \ldots, n\}$
- $\lambda \notin \mathsf{vars}(P) \cup \mathsf{vars}(Q)$
- $C[B_1, \ldots, B_n]$ *is ground*
- $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \models_{E_{\mathsf{senc}}, q_1} \mathsf{secret}(x_1, \ldots, x_n)$ *where* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$
- $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \models_{E_{\mathsf{senc}}, q_2} \mathsf{secret}(x_1, \ldots, x_n, x_s^*)$

Then $\lceil \nu k_1 \cdot \ldots \cdot \nu k_m \cdot !_w \nu\lambda \cdot (D_1^{(c,\lambda)}[B_1^{(b,\lambda)}] \mid D_2^{(c,\lambda)}[B_2^{(b,\lambda)}] \mid \ldots \mid D_n^{(c,\lambda)}[B_n^{(b,\lambda)}]) \rceil \models_{E_{\mathsf{senc}} \cup E_{\mathsf{tag}}, q_1 q_2}$ $\mathsf{secret}(x_s^*)$.

As a final result, we will show how protocols containing unbounded replication can be composed. We will use an identical syntax and semantics for contexts and processes with replication as in Section 5.7. Notice that we cannot state a result in the style of Theorem 6.3 with non-trivial probabilities. This is because, in the unbounded setting, a attacker can always amplify the probability of deriving a secret by running an attack on more sessions of a protocol. Such a restriction makes our result for unbounded processes almost identical to that of Theorem 6 from [54]. Our result, however, has two main advantages. It elimiates the still applicable attack of Example 6.3 while considering a richer class of processes.

Theorem 6.4 *Let* $C[\Box_1, \ldots, \Box_n] = \nu k_1 \cdot \ldots \cdot \nu k_m . !\nu\lambda \cdot (D_1[\Box_1] \mid D_2[\Box_2] \mid \ldots \mid D_n[\Box_n])$ *for* $w \in \mathbb{N}$ *be a context over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_c$ *and* $B_1, B_2, \ldots, B_n$ *be roles over* $\mathcal{F}_{\mathsf{senc}}$ *with labels from* $\mathcal{L}_b$ *Further let* $q_1, q_2 \in [0,1]$, $x_s \in \bigcup_{i=1}^{n} \mathsf{vars}(B_i) \setminus \mathsf{vars}(C)$ *and assume the following hold.*

- $\mathsf{fv}(C) = \emptyset$ *and* $\mathsf{fv}(B_i) \subseteq \{x_i\}$
- $\mathsf{vars}(C) \cap \mathsf{vars}(B_i) \subseteq \{x_i\}$ *for* $i \in \{1, \ldots, n\}$
- $\lambda \notin \mathsf{vars}(P) \cup \mathsf{vars}(Q)$
- $C[B_1, \ldots, B_n]$ *is ground*
- $C[\mathtt{out}(\sharp(x_1))^{\ell_1}, \ldots, \mathtt{out}(\sharp(x_n))^{\ell_n}] \models_{E_{\mathsf{senc}}, q_1} \mathsf{secret}(x_1, \ldots, x_n)$ *where* $\ell_1, \ldots, \ell_n \in \mathcal{L}_b$
- $\nu k \cdot (x_1 := k) \cdot \ldots \cdot (x_n := k) \cdot (B_1 \mid \ldots \mid B_n) \models_{E_{\mathsf{senc}}, q_2} \mathsf{secret}(x_1, \ldots, x_n, x_s^*)$

Then $\lceil \nu k_1 \cdot \ldots \cdot \nu k_m . !\nu\lambda \cdot (D_1^{(c,\lambda)}[B_1^{(b,\lambda)}] \mid D_2^{(c,\lambda)}[B_2^{(b,\lambda)}] \mid \ldots \mid D_n^{(c,\lambda)}[B_n^{(b,\lambda)}]) \rceil \models_{E_{\mathsf{senc}} \cup E_{\mathsf{tag}}, q_1 q_2} \mathsf{secret}(x_s^*)$.

CHAPTER 7: EXACT QUANTITATIVE MODEL CHECKING

In Chapter 4, we presented a technique to analyze security protocols by reducing the problem to an analysis of (belief) MDPs. In this chapter, we study the general problem of model checking probabilistic models such as DTMCs (Section 2.2) and MDPs (Section 2.3). These models have been effectively used to analyze security protocols for anonymous communication such as [64]. However, there are many other application areas such as distributed systems [107, 108], hardware communication protocols [109], reliability engineering in circuits [110, 111, 112, 113], dynamic power management [114, 115], networking [116, 117] and security [41]. Probabilistic transitions in these models are used to capture random faults, uncertainty of environment and explicit randomization used in algorithms. Analyzing properties of these probabilistic models is typically achieved through Probabilistic Computation Tree Logic (PCTL) model checking [118], wherein, desired properties of the model are specified as PCTL formulas and their validity is evaluated against the system in question.

PCTL is a quantitative extension of the temporal logic Computation Tree Logic (CTL) used to describe how a system evolves over time. For example, a PCTL formula ψ can be used to specify the property that almost surely no execution of a probabilistic program leads to a state with a deadlock. Given $\bowtie \in \{\leq, <, \geq, >\}$, the formula $\mathcal{P}_{\bowtie p}[\psi]$ expresses the property that the measure of computation paths satisfying ψ is $\bowtie p$. For a DTMC or MDP $\mathcal{M}$ and a PCTL formula ϕ, the PCTL model checking procedure recursively computes the set of states of $\mathcal{M}$ that satisfy subformulas of ϕ. Each recursive step, in turn, reduces to *constrained quantitative reachability*, wherein, given a set of good states G and a set of target states T, the goal is to compute the measure of the paths that reach T while remaining in G. If the model is decorated with *costs* or *rewards*, one may also be interested in computing the expected cost/reward of reaching T. It is well known that the constrained quantitative reachability problem for DTMCs and MDPs can be solved in polynomial time by a reduction to linear programming [119, 118].

Despite its low asymptotic complexity, linear programming, unfortunately, doesn't scale to large models and is rarely used to solve the constrained quantitative reachability problem in practice. Instead, probabilistic model checkers [61, 120, 121, 122, 62, 123], typically compute *approximations* to the exact reachability probabilities through an iterative process. The most prevalent iterative technique is *value iteration*, where exact reachability probabilities may only be approached in the limit. To ensure completion in a finite number of steps, it is common practice for model checking tools to terminate value iteration based on various

heuristics, for example, when the difference between the computed reachability probabilities of successive iterations is "small". This approximation step may lead to unsound results [124, 125, 126], particularly in systems where high magnitude changes in value iteration are preceded by periods of stability that cause iteration to terminate prematurely.

Another iterative technique for computing constrained quantitative reachability is *interval iteration* [125, 124, 127]. Aimed at addressing the shortcomings of value iteration, interval iteration utilizes two simultaneous value iteration procedures converging to the exact probability values from above and below. While this allows one to bound the error present in the approximation, the exact solution cannot be obtained from such an interval bound. Further, state-of-the-art model checkers typically implement these iterative procedures using floating-point numbers and finite- precision arithmetic. As a result, both iterative techniques are susceptible to overflows in floating-point calculations. The inherent imprecision in the approximate answers, combined with the errors introduced from finite precision arithmetic can be further compounded by the presence of nested probability operators in PCTL formulas when the sets of good states G and target states T are not correctly computed in the recursive step (see Example 7.3 in Section 7.2).

We present a new algorithm and its implementation that *sharpens* approximate solutions computed by fast iterative techniques, to obtain the *exact* constrained reachability probabilities. The starting point of our approach is the observation that when transition probabilities in the model are rational numbers, the exact solution is also a rational number of polynomially many bits. The second ingredient in our technique is an algorithm due to Kwek and Mehlhorn [128], which, given a "close enough" approximation to a rational number, finds the rational number efficiently. The rough outline of our algorithm is as follows. We use an iterative technique (value iteration or interval iteration) to compute an approximate solution and then apply the Kwek-Mehlhorn algorithm to find a close candidate rational solution. Since the approximate solution we start with is of unknown quality, the candidate rational solution obtained may not be the exact answer. Therefore, we check if the candidate is the unique solution to the linear program that describes the system. This allows one to confirm the correctness of the candidate rational solution. If it is not correct, the process is repeated, starting with an approximate solution of improved precision. Precise details of the algorithm are given in Section 7.3.

We have implemented this approach as an extension of the PRISM model checker, called RationalSearch. Our tool computes exact constrained reachability probabilities and exact expected rewards when model checking DTMCs and MDPs against PCTL specifications. Evaluation of our implementation against a large set of examples from the PRISM benchmark suite [129] and case studies [130] shows that our technique can be applied to a wide

array of examples. In many cases, our tool is orders of magnitude faster than the exact model checking engines implemented in state-of-the-art tools like PRISM [61] and STORM [62].

The work closest in spirit to ours is [131], which presents an approach to obtain exact solutions for reachability properties for MDPs and discounted MDPs. The basic idea there is to interpret the scheduler obtained for an approximate solution, as a *basis* for the linear program corresponding to the verification question. By examining the optimality of the solution associated with this basis, the exact solution can be obtained by improving the scheduler using the Simplex algorithm. This is significantly different from our approach. In particular, for DTMCs (where there is no scheduler), the approach of [131] reduces to solving a linear program, which is known to be not scalable. Since the implementation from [131] is not available, we could not experimentally compare with this approach.

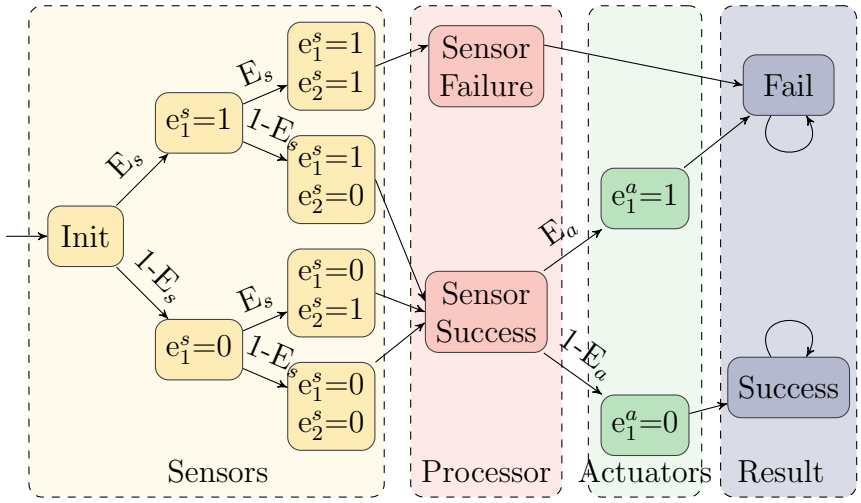
Several existing tools [62, 61] implement algorithms for exact quantitative model checking. Essentially these tools work by creating a model representation using rational numbers and performing a state elimination computation similar to Gauss elimination. Much of the infrastructure of this computation can be derived from *parametric model checking* techniques [132, 120, 133, 134] that analyze systems in which portions of the model are left unspecified. These computations are intrinsically more complicated than those performed by approximation engines. Our techniques avoid these expensive computations while still producing exact solutions for a large class of examples.

7.1 PROBABILISTIC COMPUTATION TREE LOGIC

In Section 2.2, DTMCs were defined as a tuple (Z, z_s, Δ) where Z is a finite set of states, $\Delta : Z \to \mathsf{Dist}(Z)$ is the probabilistic transition function and z_s is the start state. In this chapter, we will not assume a start state z_s and will additionally enrich DTMCs with a cost (or reward) structure $\mathbf{C} : Z \times Z \mapsto \mathbb{Q}^{\geq 0}$ and a labeling function $L : Z \to 2^{\mathsf{AP}}$ that maps states to subsets of AP, the set of atomic propositions. We will henceforth consider a DTMC to be a tuple $\mathcal{M} = (Z, \Delta, \mathbf{C}, L)$. We will write $\mathsf{Exec}_z^{\infty}(\mathcal{M})$ to be the executions of $\mathcal{M}$ stating from state z. The measure $\mathsf{prob}_{\mathcal{M}}$ is defined exactly as it is in Section 2.2. Let $z \in Z$ and $F \subseteq Z$. Define a function $\mathsf{cost}_z(F, \mathcal{M}) : \mathsf{Exec}_z^{\infty}(\mathcal{M}) \to \mathbb{Q}^{\geq 0}$ such that for any $\rho \in \mathsf{Exec}_z^{\infty}(\mathcal{M})$, $\mathsf{cost}_z(F, \mathcal{M})(\rho) = \sum_{i=0}^{m-1} \mathbf{C}(z_i, z_{i+1})$ if $z_0 \to \cdots \to z_m$ is the shortest prefix of ρ such that $z_m \in F$ and $\mathsf{cost}_z(F, \mathcal{M})(\rho) = \infty$ if no such prefix exists. Let E_z be the usual expectation on $\mathsf{Exec}_z(\mathcal{M})$ with respect to the measure $\mathsf{prob}_{\mathcal{M}}$. Then $\mathsf{E}_z[\mathsf{cost}_z(F, \mathcal{M})]$ is the expected cost of reaching F.

Example 7.1 *Consider an embedded control system [135] comprised of an input processor,*

Figure 7.1 *Markov chain for a simple embedded control system with two sensors and one actuator tolerating a single sensor fault.*



a main processor, an output processor and a bus. In each cycle of the system, the input processor collects data from a set of n sensors $S_1, S_2, \ldots, S_n$. *The main processor polls the input processor and passes instructions to the output processor controlling a set of m actuators* $A_1, A_2, \ldots A_m$. *Communication between processors occurs over the bus. The system is designed to tolerate failures in a limited number of components. If the input processor reports that the number of sensor failures exceeds some threshold* MAX_FAILURES, *then the main processor shuts the system down. Otherwise, it activates the actuators, which again, are prone to failure. When the probabilities with which each of these components fail are known, one can model the system's reliability using a DTMC. Figure 7.1 shows a DTMC that models a single cycle of such a system with* $n = 2$ *sensors and* $m = 1$ *actuator. For simplicity, we assume that each sensor fails with probability* E_s *and each actuator fails with probability* E_a. *States of the model are labeled with* $e_1^s, \ldots, e_n^s \in \{0, 1\}$ *and* $e_1^a, \ldots, e_m^a \in \{0, 1\}$, *where* $e_i^s = 1$ *denotes the failure of sensor* S_i *and* $e_i^a = 1$ *denotes the failure of actuator* A_i. *In Figure 7.1, we omit labels if they are not relevant in a particular state.*

As was done for DTMCs, we will also augment MDPs with a cost (or reward) structure $\mathbf{C} : Z \times \mathsf{Act} \times Z \to \mathbb{Q}^{\geq 0}$ and a labeling function $L : Z \to 2^{\mathsf{AP}}$ such that an MDP is a tuple $\mathcal{M} = (Z, \mathsf{Act}, \Delta, \mathbf{C}, L)$. One can define the expected cost of reaching a target set of states in a similar fashion to the DTMC case. Interested readers should refer to standard texts such as [136, 119] for more details. Properties of DTMCs and MDPs can be expressed in the logic PCTL, which extends the temporal logic CTL with the ability to reason quantitatively.

Definition 7.1 *Let $a \in \mathsf{AP}$ be an atomic proposition, $\bowtie \in \{\leq, <, \geq, >\}$, $p \in [0,1]$, $c \in \mathbb{Q}^{\geq 0}$ and $k \in \mathbb{N}$. The syntax of PCTL is*

$$\phi ::= \mathsf{true} \;\Big|\; a \;\Big|\; \neg\phi \;\Big|\; \phi \wedge \phi \;\Big|\; \mathcal{P}_{\bowtie p}[\psi] \;\Big|\; \mathcal{E}_{\bowtie c}[\phi]$$

where $\psi ::= \mathcal{X}\phi \mid \phi \mathcal{U} \phi$.

In Definition 7.1, ϕ is a state formula used to describe properties of states and ψ are path formulas used to model properties of paths. We now formalize the semantics of PCTL.

Definition 7.2 *Let $\mathcal{M} = (Z, \Delta, \boldsymbol{C}, L)$ be a DTMC, ϕ, ϕ_1, ϕ_2 be state formulas and ψ be a path formula. The satisfaction relation $\models$ for PCTL state formulae is defined inductively as*

$$\begin{aligned}
z &\models \mathsf{true} && \text{for all } z \in Z \\
z &\models a && \Leftrightarrow a \in L(z) \\
z &\models \neg\phi && \Leftrightarrow z \not\models \phi \\
z &\models \phi_1 \wedge \phi_2 && \Leftrightarrow z \models \phi_1 \text{ and } z \models \phi_2 \\
z &\models \mathcal{P}_{\bowtie p}[\psi] && \Leftrightarrow p_z(\psi) \bowtie p \\
z &\models \mathcal{E}_{\bowtie c}[\phi] && \Leftrightarrow e_z(\phi) \bowtie c
\end{aligned}$$

where $p_z(\psi) = \mathsf{prob}(\{\rho \in \mathsf{Exec}_z(\mathcal{M}) \mid \rho \models \psi\})$, $e_z(\phi) = \mathsf{E}[\mathsf{cost}_z(Z_\phi)]$ with $Z_\phi = \{z' \in Z \mid z' \models \phi\}$, and the satisfaction relation for paths and path formulae is defined inductively as

$$\begin{aligned}
\rho &\models \mathcal{X}\phi && \Leftrightarrow \rho(1) \models \phi \\
\rho &\models \phi_1 \mathcal{U} \phi_2 && \Leftrightarrow \exists i \geq 0 : (\rho(i) \models \phi_2 \;\mathscr{E}\; \forall j < i : \rho(j) \models \phi_1)
\end{aligned}$$

When the underlying model $\mathcal{M}$ is an MDP, the semantics of PCTL formulae stay the same, except for the semantics of $\mathcal{P}_{\bowtie p}[\psi]$ and $\mathcal{E}_{\bowtie c}[\phi]$, which now require a quantification over all schedulers. Let $p_z^{\mathcal{A}}(\psi) = \mathsf{prob}(\{\rho \in \mathsf{Exec}_z(\mathcal{M}^{\mathcal{A}}) \mid \rho \models \psi\})$. One can analogously define $e_z^{\mathcal{A}}(\phi)$ for a scheduler $\mathcal{A}$.

Definition 7.3 *Let $\mathcal{M}$ be an MDP, ϕ be a state formula and ψ be a path formula. The satisfaction relation $\models$ for PCTL state formulae is defined identically to Definition 7.2, with the exception of the following cases.*

$$\begin{aligned}
z &\models \mathcal{P}_{\bowtie p}[\psi] && \Leftrightarrow \forall \mathcal{A} \in \mathcal{S},\, p_z^{\mathcal{A}}(\psi) \bowtie p \\
z &\models \mathcal{E}_{\bowtie c}[\phi] && \Leftrightarrow \forall \mathcal{A} \in \mathcal{S},\, e_z^{\mathcal{A}}(\phi) \bowtie c
\end{aligned}$$

For a path formula ψ (resp. state formula ϕ), we write $\mathcal{P}_{=?}[\psi]$ (resp. $\mathcal{E}_{=?}[\phi]$) to represent the solution vector V, given by $\mathsf{V}(z) = p_z(\psi)$ (resp. $e_z(\phi)$) for all $z \in Z$. Strictly speaking, $\mathcal{P}_{=?}[\cdot]$ and $\mathcal{E}_{=?}[\cdot]$ are not part of PCTL syntax. However, we henceforth extend the PCTL syntax to allow $\mathcal{P}_{=?}[\cdot]$ and $\mathcal{E}_{=?}[\cdot]$ as the outermost operator.

Example 7.2 *Consider the DTMC modeling an embedded control system from Example 7.1. One can describe many important properties of this model using PCTL.*

1. *The probability of success:*

$$\mathcal{P}_{=?}\ [\ \mathsf{true}\,\mathcal{U}\ \text{“}Sucess\text{”}\ ]$$

2. *The probability of reaching states where there are no sensor failures from a given state:*

$$\mathcal{P}_{=?}\ [\ \mathsf{true}\,\mathcal{U}\ (e_1^s + ... + e_n^s = 0)\ ]$$

3. *Let G be the set of states where the probability of reaching a state in which sensor S_1 fails is $\bowtie 1/2$. Let T be the set of states for which the probability of reaching a state in which actuator A_1 fails is 0. The probability of remaining in states from G until reaching a state from T:*

$$\mathcal{P}_{=?}\ [\ \mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\,\mathcal{U}\ (e_1^s{=}1)]\ \mathcal{U}\ \mathcal{P}_{\leq 0}[\mathsf{true}\,\mathcal{U}\ (e_1^a{=}1)]\ ]$$

Similar to the model checking algorithm for CTL, the PCTL model checking algorithm recursively computes the set of states satisfying a state sub-formula (see [136, 119] for the complete details). We will begin by restricting our attention to DTMCs. Let ϕ, ϕ' be state formulas. To compute $\mathcal{P}_{=?}[\phi\ \mathcal{U}\ \phi']$, one recursively computes the set of states Z_ϕ and $Z_{\phi'}$ satisfying ϕ and ϕ', respectively. These can be used to derive, for every $z{\in}Z$, the quantity $p_z(\phi\ \mathcal{U}\ \phi')$ which represents the probability of reaching the set $Z_{\phi'}$ while remaining in the set Z_ϕ, starting from the state z. The probability $p_z(\phi\,\mathcal{U}\,\phi')$ can be computed as the unique solution to the following linear program.

$$p_z(\phi\,\mathcal{U}\,\phi') = \begin{cases} 0 & \text{if } z \in \mathsf{Prob0} \\ 1 & \text{if } z \in \mathsf{Prob1} \\ \sum_{z' \in Z} \Delta(z,z') \cdot p_{z'}(\phi\,\mathcal{U}\,\phi') & \text{otherwise} \end{cases} \tag{7.1}$$

In the equation above, $\mathsf{Prob0}$ and $\mathsf{Prob1}$ are the set of states that satisfy $\phi\,\mathcal{U}\phi'$ with probability 0 and 1, respectively. These sets can be determined via a pre-computation step that analyzes the underlying graph of the DTMC. To compute $\mathcal{P}_{\bowtie p}[\phi\,\mathcal{U}\,\phi']$, one computes $\mathcal{P}_{=?}[\phi\,\mathcal{U}\,\phi']$ and does the comparison $p_z(z\,\mathcal{U}\,z') \bowtie p$ for every $z \in Z$. The computations for $\neg\phi$, $\phi \wedge \phi'$, $\mathcal{E}_{=?}[\phi]$ and $\mathcal{P}_{\bowtie c}[\mathcal{X}\phi]$ are similar.

When the underlying model is an MDP, the computation for $\mathcal{P}_{\bowtie p}[\phi\,\mathcal{U}\,\phi']$ reduces to solving

the following linear optimization problem when $\bowtie \in \{<, \leq\}$

$$\begin{aligned} &\min \sum_{z \in Z} p_z(\phi \,\mathcal{U}\, \phi') \quad \text{subject to} \\ &p_z(\phi \,\mathcal{U}\, \phi') = 0 && \text{if } z \in \mathsf{Prob0} \\ &p_z(\phi \,\mathcal{U}\, \phi') = 1 && \text{if } z \in \mathsf{Prob1} \\ &p_z(\phi \,\mathcal{U}\, \phi') \geq \sum_{z' \in Z} \Delta(z, \alpha, z') \cdot p_{z'}(\phi \,\mathcal{U}\, \phi') \\ &\qquad \text{for each } \alpha \in \mathsf{enabled}(z) && \text{otherwise} \end{aligned} \tag{7.2}$$

When $\bowtie \in \{>, \geq\}$, the objective changes to maximization and the direction the last inequality is reversed.

7.2 APPROXIMATE MODEL CHECKING

As discussed above, solving quantitative properties of DTMCs and MDPs by a reduction to linear programming does not scale well enough to make it a viable solution technique in practice. As a result, techniques to approximate solutions using floating point arithmetic have been widely adopted. In this section, we describe two such techniques, value iteration and interval iteration and demonstrate how each approach can produce incorrect solutions.

7.2.1 Iterative Techniques

The linear program described in equation (7.1) for DTMCs can equivalently be expressed in the form of equation (7.3) below, for some appropriate matrix A and vector b.

$$\bar{x} = A\bar{x} + b \tag{7.3}$$

This allows for an alternate approach to solving the linear program from equation (7.1) known as *value iteration*. In the case of DTMCs, one iteratively computes the solution vector as the limit of the sequence $\{\bar{x}_i\}_{i \geq 0}$ given by $\bar{x}_{i+1} = A\bar{x}_i + \bar{b}$ starting with $\bar{x}_0$ where

$$\bar{x}_0(z) = \begin{cases} 1 & \text{if } z \in \mathsf{Prob1} \\ 0 & \text{otherwise .} \end{cases}$$

The linear program from equation (7.2) for MDPs can likewise be expressed in the form of equation (7.4) below.

$$\bar{x}(z) = \max\{\sum_{z' \in Z} \Delta(z, \alpha, z') \cdot \bar{x}(z') \mid \alpha \in \mathsf{enabled}(z)\} \tag{7.4}$$

In this case again, the exact solution can be expressed as the limit of an iterative sequence $\{x_i\}_{i \geq 0}$. In many cases, the sequence does not converge in a finite number of steps, and therefore model checkers terminate the sequence when successive vectors v_k and v_{k+1} become "close enough". The choice of stopping criterion is based largely on heuristics. The PRISM model checker, for example, implements two criteria (i) *absolute convergence*, and (ii) *relative convergence*. Under the absolute criterion, value iteration terminates if the norm $\|v_{k+1} - v_k\| < \epsilon$ for some $\epsilon > 0$. Under the relative criterion, termination occurs when $\frac{\|v_{k+1} - v_k\|}{\|v_k\|} < \epsilon$. In spite of the fact that iterative techniques only approximate solutions, value iteration remains the popular choice for widely used tools that analyze PCTL properties as it vastly outperforms linear programming techniques, despite their theoretically better asymptotic complexity.

As originally observed in [137], value iteration provides no guarantees about the quality of the solution, regardless of the stopping criterion used. To help rectify this problem, Haddad et. al. [125] introduce *interval iteration* for computing min/max reachability probabilities in DTMCs and MDPs. In this approach, one simultaneously computes two sequences of vectors, one converging to the solution from below and one converging to the exact solution from above. In this setting, the stopping criterion becomes straightforward; terminate when the distance between the two vectors is within some ϵ threshold. Assuming the absence of floating point errors, this effectively gives a small ϵ-neighborhood that contains the actual solution. In order to achieve convergence, interval iteration requires a pre-processing step that transforms the underlying graph of the model. The interval iteration technique was extended to costs (rewards) in [124].

Both iterative techniques described above can be further enhanced by performing arithmetic operations using *Multi-terminal binary decision diagrams* (MTBDDs) [138, 139]. MTBDDs generalize BDDs [140] by allowing terminal values to be different from 0 or 1. Similar to the role of BDDs in symbolic model checking [141], MTBDD based model checkers leverage the performance benefit due to the succinct representations of the data structures involved. Let $\mathsf{vars} = \{v_1, v_2, \ldots v_k\}$ be a finite and ordered set of boolean variables, and let $\mathcal{D}$ be some domain of values. Given a function $f : 2^{\mathsf{vars}} \rightarrow \mathcal{D}$, an MTBDD that represents f is a full binary tree of height $|\mathsf{vars}|$ with leaf nodes labeled with elements from $\mathcal{D}$ and internal nodes at depth i labeled with variable v_{i+1}. Each path in an MTBDD then represents a specific valuation of the variables vars and the leaf node represents the value of f for this valuation. A *reduced order* MTBDD, similar to a reduced order BDD (ROBDD), merges isomorphic

subtrees in the MTBDDs. In particular, this means that, a reduced order MTBDD has exactly one leaf node for every value d in the range of the function f. In what follows, we will refer reduced order MTBDDs as simply MTBDDs.

7.2.2 Shortcomings of iterative techniques

When computing constrained reachability probabilities using value iteration, both the absolute and relative convergence criteria can result in solutions that are very far from the actual answers. In [125], the authors give a DTMC and a PCTL property whose solution is $\frac{1}{2}$, yet PRISM reports 9.77×10^{-4} for the absolute criterion and 0.198 for the relative criterion. This drastic error is the result of a premature termination of value iteration. Several other sources of imprecision can also cause state-of-the-art quantitative model checkers to produce unsound results. For example, consider a PCTL formula of the form $\mathcal{P}_{\geq p}(\psi)$ and a system $\mathcal{M}$ such that the probability measure of the formula ψ is exactly p. When value iteration, with floating point numbers, is used to compute this measure, the value p may only be approached in the limit, and hence the procedure will return some p' that approximates p from below. This means that the formula $\mathcal{P}_{\geq p}(\psi)$ will evaluate to false, where of course the correct value is true. This phenomenon was first pointed out in [126]. We also demonstrate a similar phenomenon with the DTMC from Example 7.1. For the sake of illustration, let $\mathrm{E}_s = \frac{1}{2}$. Clearly, from the initial state, the probability of reaching a state where sensor 1 fails is exactly $\frac{1}{2}$ and hence the formula $\mathcal{P}_{<\frac{1}{2}}$ [$\mathsf{true}\ \mathcal{U}\ (e_1^s{=}1)$] evaluates to false for the initial state. However, PRISM returns true. Errors such as these can be compounded in PCTL formulas containing nested operators, wherein the recursive step of the model checking algorithm returns an incorrect set of states. This can lead to substantial logical errors in model analysis, as we demonstrate with the example below.

Example 7.3 *Let us instantiate the DTMC from Example 7.1 with $n = 14$ sensors, $m = 1$ actuator,* MAX_FAILURES=1 *and with $E_s = E_a = \frac{1}{2}$. Recall the third PCTL property of the embedded control system given in Example 7.2:*

$$\mathcal{P}_{=?}\ [\ \mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\,\mathcal{U}\ (e_1^s{=}1)]\ \mathcal{U}\ \mathcal{P}_{\leq 0}[\mathsf{true}\,\mathcal{U}\ (e_1^a{=}1)]\ ].$$

When $\bowtie$ is $\leq$, the PRISM model checker returns "0.7096993582589287" *as the probability for the initial state with both value iteration and interval iteration*[1]*. With our tool* RationalSearch*, one can verify that the correct probability is* 212895/229376*, or* "0.9281485421316964"*. Further, when $\bowtie$ is $<$, PRISM again returns the value given above*

[1]Using the Hybrid engine, the absolute convergence criterion and $\epsilon = 10^{-16}$.

for both iterative techniques. This time, the actual solution, as generated by RationalSearch*, is* 0*. The errors above are the result of the fact that PRISM incorrectly computes the set of states satisfying* $\mathcal{P}_{\bowtie \frac{1}{2}}[\mathsf{true}\,\mathcal{U}\,(e_1^s{=}1)]$*. This error in the recursive step results in an incorrect formulation of the constraints in the outermost constrained reachability problem.*

7.3 EXACT MODEL CHECKING

As demonstrated in the previous section, approximate solution techniques can lead to unreliable results and the incorrect analysis of systems. To rectify this serious limitation, tools such as PRISM and STORM have implemented exact model checking engines, which make heavy use of techniques from parametric model checking [132, 120, 133, 134]. The idea behind these engines is to interpret the probabilistic model (DTMC or MDP) as a finite automaton in which transitions probabilities are described by letters of an alphabet. When one is interested in cost (rewards), states are additionally labeled by a cost structure. Using techniques derived from state elimination [142], one can then calculate a regular expression representing the language of this automaton. The core idea of this translation is to eliminate a state s by increasing the probability of moving from each predecessor s_1 of s to each successor s_2 of s by the probability of moving from s_1 to s_2 when passing through s. In the case of parametric model checking, various techniques can then be used to translate the regular expression into a rational function over the parameters of the model. When using this approach for exact model checking, one can likewise derive a parameter-free function that describes the property in question.

Although they rectify the problems with approximation techniques, the exact quantitative model checking engines implemented in tools like PRISM and STORM don't scale as well as their iterative counterparts. See Example 7.4 below and Section 7.5 for a complete analysis. The goal of our technique, to which the remainder of this section is dedicated, is to utilize the advantages of fast approximate model checking techniques to produce exact solutions.

Example 7.4 *Again consider the DTMC modeling an embedded control system with the parameters given in Example 7.3. To guarantee the correctness of one's analysis, exact solution techniques must be employed. Unfortunately, the exact model checking engines of PRISM and STORM do not scale well enough to analyze this example, which contains about* 4.8 *million states and about* 44 *million transitions. Under our test setup (see Section 7.5), both tools reached a 30 minute timeout when trying to analyze the properties from Example 7.3. On the other hand,* RationalSearch *found the exact answer to both the formulae in*

under a minute.

7.3.1 The Kwek-Mehlhorn algorithm

Given an ordered set of integers of bounded size, the classical binary search algorithm can be used to locate the smallest element larger than a given value. Kwek and Mehlhorn [128] extend this methodology to efficiently locate the rational number with the smallest size in a given interval. Here we present a novel application of this technique where approximate answers to quantitative model checking problems can be used to efficiently generate exact solutions.

Consider an interval $I = [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$ with rational end-points. It was established [128] that for any interval $I = [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$, there exists a unique rational $a_{\min}(I)/b_{\min}(I)$ such that for all rational numbers $\frac{a}{b} \in I$, $a_{\min}(I) \leq a$ and $b_{\min}(I) \leq b$. Further, this minimal fraction $a_{\min}(I)/b_{\min}(I)$ can be found using Algorithm 7.1 from [128].

Algorithm 7.1 *Compute the minimal rational in* $[\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$.

function FINDFRACTION(α, β, γ, δ):
  if $\lfloor \frac{\alpha}{\beta} \rfloor = \lfloor \frac{\gamma}{\delta} \rfloor$ and $\frac{\alpha}{\beta} \notin \mathbb{N}$ **then**
    $b, a \leftarrow$ FINDFRACTION(δ, $\gamma \bmod \delta$, β, $\alpha \bmod \beta$)
    return $\lfloor \frac{\alpha}{\beta} \rfloor b + a, b$
  else
    return $\lceil \frac{\alpha}{\beta} \rceil, 1$

Let $Q_M = \{p/q \mid p, q \in \{1, \ldots, M\}\} \cap [0, 1]$. For $\mu \in \mathbb{N}$, if $\frac{a}{b} \in Q_M$ is contained in the interval $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$ of length $\frac{1}{2M^2}$ then $\frac{a}{b}$ is the unique element of Q_M in $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$. It turns out that $\frac{a}{b}$ must also be the minimal element of $[\frac{\mu}{2M^2}, \frac{\mu+1}{2M^2}]$, meaning it can be found using Algorithm 7.1 in time $O(\log M)$.

7.3.2 Rational search

In this section, we explain our approach for exact quantitative model checking of PCTL formulas. The key insight we exploit is that iterative techniques for solving constrained reachability typically converge very fast and produce a precise enough answer. Using this precise approximation, we can then effectively construct a small interval for which the Kwek-Mehlhorn algorithm can find the exact solution.

Recall that each iterative technique for approximating a set of equations, like those given in equations (7.1) and (7.2), yields a different guarantee on the precision of an approximate

solution. The difference between the approximation generated by interval iteration and the actual solution is bounded by a given ϵ value, provided there are no errors generated by floating-point arithmetic. Value iteration, on the other hand, comes with no guarantees. When an approximate solution vector contains values of known precision, one can translate it into an exact solution vector as follows. For each value q in the vector, construct the interval $[q - \epsilon, q + \epsilon]$ and run Algorithm 7.1 to find the smallest rational in this interval. Then, check that the generated rational values $\mathsf{V}^\star$ are correct by verifying that they satisfy equation (7.3) or (7.4), whichever is appropriate. The uniqueness of the solutions to the these equation systems (which follows from those of (7.1) and (7.2)) ensures that the fixpoint check is only satisfied by the desired solution vector. If the fixpoint check fails for the candidate solution vector, one obtains a more precise approximation and re-runs the procedure.

When a solution vector contains values of unknown quality, we can find exact solutions using a similar technique. Here the idea is to "guess" a sequence intervals, with decreasing sizes, that may contain the actual value. This process is formalized in Algorithm 7.2, which takes as input the model $\mathcal{M}$, a maximum precision P and a state-indexed vector $\mathsf{V}^\dagger$ that approximates the exact solution vector V.

Algorithm 7.2 *Sharpen values of unknown precision.*

```
function SHARPEN(M, P, V†):
    for all p ∈ {1, ..., P} do
        for all z ∈ Z do
            α, β, γ, δ ← BOUNDS(p, V†(z))
            V⋆(z) ← ⌊V†(z)⌋ + FINDFRACTION(α, β, γ, δ)
        if FIXPOINT(M, V⋆) then
            return V⋆
    return null
```

For a given precision p and state z, $\textsc{bounds}(p, \mathsf{V}^\dagger(z))$ returns $\alpha, \beta, \gamma, \delta$ such that α is the first p decimal digits of the fractional part of $\mathsf{V}^\dagger(z)$, $\beta = 10^p$, $\gamma = \alpha + 1$ and $\delta = \beta$. Observe that α/β is the rational representation of the first p digits of the fractional part of $\mathsf{V}^\dagger(z)$. From this approximation, we identify a *sharpened* solution vector $\mathsf{V}^\star$ using the FINDFRACTION procedure from Algorithm 7.1. The procedure FIXPOINT then tests if $\mathsf{V}^\star$ is the correct solution by checking if it satisfies (7.3) or (7.4). If the input vector $\mathsf{V}^\dagger$ is not precise enough, then SHAPREN returns "null", indicating that more precision is required to infer an exact solution. The guarantees of Algorithm 7.2 are formalized as follows. Let V^b satisfying $\mathsf{V}(z) - \mathsf{V}^b(z) \leq 10^{-b}$ for all $z \in Z$ be an approximate solution vector of precision b. Then, Lemma 7.1 establishes that starting from a close enough approximation, Algorithm 7.2 finds

the actual solution vector.

Lemma 7.1 *Let $\mathcal{M}$ be an MDP, ψ be a PCTL path formula and V be the solution vector for $\mathcal{P}_{=?}[\psi]$. Let $b, P \in \mathbb{N}$ be such that $P \geq b$ and V^b is an approximate solution vector of precision b. If $\mathsf{V}(z) \in Q_{\lfloor\sqrt{10^b/2}\rfloor}$ for every $z \in Z$, then* $\textsc{sharpen}(\mathcal{M}, P, \mathsf{V}^b) = \mathsf{V}$.

Proof. Fix a state z and assume $\mathsf{V}(z) \in Q_M$ for $M{=}\lfloor\sqrt{10^b/2}\rfloor$. If $P{\geq}b$ then $\textsc{sharpen}(\mathcal{M}, P, \mathsf{V}^b)$ searches for $\mathsf{V}(z)$ in $I = [\alpha/\beta, \gamma/\delta]$ for $\alpha, \beta, \gamma, \delta = \textsc{bounds}(b, \mathsf{V}^b(z))$. Now, $\mathsf{V}(z) \in I$ since $\mathsf{V}^b(z)$ satisfies $\mathsf{V}(z) - \mathsf{V}^b(z) \leq 10^{-b}$. Further, $|I| = 10^{-b} \leq \frac{1}{2M^2}$. Due to Kwek et. al. [128], we have that an interval of size $\frac{1}{2M^2}$ contains at most 1 element of Q_M. Clearly, $\textsc{findFraction}(\alpha, \beta, \gamma, \delta)$ returns $\mathsf{V}(z)$ which is the unique "minimal" element in $I \cap Q_M$.

We now describe a method for optimizing Algorithm 7.2. Let v be a floating-point value and p be a precision. If the decimal expansion of $\textsc{findFraction}(\textsc{bounds}(p, v))$ agrees with v on the first q digits, then

$$\textsc{findFraction}(\textsc{bounds}(p, v)) = \textsc{findFraction}(\textsc{bounds}(p', v))$$

for all $p' \leq q$. The correctness of this observation follows from Lemma 7.2 below.

Lemma 7.2 *Let $\alpha, \beta, \gamma, \delta$ and $\alpha', \beta', \gamma', \delta'$ be integers such that the following hold.*

1. $[\frac{\alpha'}{\beta'}, \frac{\gamma'}{\delta'}] \subseteq [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$
2. $\textsc{findFraction}(\alpha, \beta, \gamma, \delta) \in [\frac{\alpha'}{\beta'}, \frac{\gamma'}{\delta'}]$

Then $\textsc{findFraction}(\alpha, \beta, \gamma, \delta) = \textsc{findFraction}(\alpha', \beta', \gamma', \delta')$.

Using the techniques for sharpening an approximate solution into an exact value from Algorithm 7.2, we can now derive a procedure for solving constrained reachability (and hence PCTL) formulas exactly. The procedure is given in Algorithm 7.3 which takes as arguments an MDP or DTMC $\mathcal{M}$, a constrained reachability formula ϕ and a precision ϵ. The ITERATION procedure can be either of value iteration or interval iteration. Algorithm 7.3 begins by running the iteration procedure up to a given precision ϵ. If the procedure is value iteration, ϵ is used in the convergence criterion — absolute or relative — described in Section 7.2. In the case of interval iteration, ϵ defines the bound on the maximum error in the approximate solution vector. The approximate solution vector $\mathsf{V}^\dagger$ generated by the iteration procedure is then used by the SHAPREN procedure, which attempts to strengthen

the approximate answer to an exact one. Note the the version of the SHAPREN varies according to iterative method being utilized. If it succeeds, the whole process terminates. Otherwise, $\mathsf{V}^\dagger$ is further refined by re-invoking ITERATION with an increased ϵ precision and the sharpening process is repeated.

Algorithm 7.3 *Rational search.*

function RATIONALSEARCH($\mathcal{M}$, ϕ, ϵ_0):
  $\mathsf{V}^{\text{init}} \leftarrow$ INIT($\mathcal{M}, \phi$)
  $\epsilon \leftarrow \epsilon_0$
  while true **do**
    $\mathsf{V}^\dagger \leftarrow$ ITERATION($\mathcal{M}$, ϕ, V^{init}, ϵ)
    $\mathsf{V}^\star \leftarrow$ SHARPEN($\mathcal{M}$, $\lceil \log(\frac{1}{\epsilon}) \rceil$, $\mathsf{V}^\dagger$)
    if $\mathsf{V}^\star \neq$ null **then**
      return $\mathsf{V}^\star$
    $\mathsf{V}^{\text{init}} \leftarrow \mathsf{V}^\dagger$
    $\epsilon \leftarrow \epsilon/10$

When successive approximations in value iteration or interval iteration are computed using arbitrary precision arithmetic, the correctness guarantees of Algorithm 7.3 can be stated as follows.

Theorem 7.1 *Let $\mathcal{M}$ be a MDP, ψ be a constrained reachability formula and V be the solution vector for $\mathcal{P}_{=?}[\psi]$ and $\epsilon_0 \in \mathbb{Q}^{>0}$. Then,* RATIONALSEARCH($\mathcal{M}, \mathcal{P}_{\bowtie p}[\psi], \epsilon_0$) *terminates and returns the exact solution vector V.*

Proof. It is easy to see that there is a $b > 0$ such that, for every state z, $\mathsf{V}(z) \in Q_N$ for $N = \lfloor \sqrt{10^b/2} \rfloor$. Now, since value iteration converges in the limit, we have that the first b digits of $\mathsf{V}^\dagger(z)$ match that of $\mathsf{V}(z)$ for each state $z \in Z$, eventually. Also, in every iteration of the loop in Algorithm 7.3, SHARPEN is invoked with an incremented value of P and eventually $P \geq b$.

While both Lemma 7.1 and Theorem 7.1 are stated for formulae of the kind $\mathcal{P}_{=?}[\psi]$, they can be easily re-factored to reason about formulas of the form $\mathcal{E}_{=?}[\phi]$.

Example 7.5 *Our experiments show that Algorithm 7.3 can make non-trivial improvements to solution quality. Consider the standard example of tossing N biased coins independently, where each coin yields heads with probability $1/3$ and tails with probability $2/3$. Analyzing the DTMC model to compute the probability of the event that 11 coins land heads, PRISM's floating-point model checker returned the decimal "0.0000056450292694767 58". Our tool was able to correctly determine the exact probability to be 1/177,147 by examining with the first 12 digits of this approximate answer. This is remarkable given that the period of this*

fraction (and hence its most succinct decimal representation) is almost 20,000 digits long. Moreover, the algorithm is able to simultaneously infer the reachability probabilities for all *of the roughly 200,000 states of the model with a single fixpoint check. This illustrates another advantage of our technique; the algorithm is agnostic of the number of initial states in the system. The exact model checking engine of PRISM, on the other hand, currently only supports systems with a single initial state.*

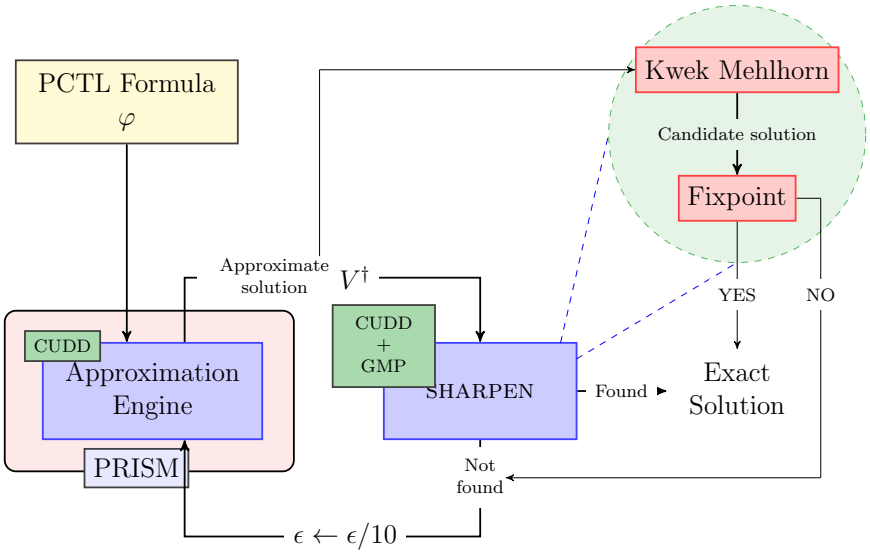
7.4 IMPLEMENTATION

We have implemented Algorithm 7.3 in our tool RationalSearch, which is an extension of the PRISM model checker (version 4.3.1). RationalSearch is available for download at [143]. Before describing our integration with PRISM, we briefly describe the relevant portions of its architecture. PRISM is a Java-based tool comprised of four solution engines, three of which (Mtbdd, Hybrid, Sparse) are based (entirely or partially) on symbolic methods using compact data structures like MTBDDs. The fourth engine (Explicit) manipulates sparse matrices, vectors and bit-sets directly.

The Sparse engine is similar to the Explicit engine in that it uses explicit data structures for storing vectors and matrices. However, it makes use of symbolic data structures during model construction, allowing it to efficiently remove portions of the state space that are not reachable. This is achieved through a conjunction of the MTBDD representing the model's state space with a BDD representing the characteristic function for the reachable states of the model. The Mtbdd engine is based entirely on symbolic data structures. During value iteration, the transition matrix and solution vector are both given as MTBDDs. The matrix-vector multiplications used to update the solution vector are carried out over these data structures. As described in [144] one drawback of this approach is that the size of the solution vector can grow substantially as more computations are performed. To address this issue, the Hybrid engine combines the advantages present in both the symbolic and explicit engines. In particular, it stores the solution vector as a fixed size array and the transition matrix as an MTBDD (which can usually be done succinctly due to symmetry in the model). Updates to the solution vector are carried out by operations over these mixed-type data structures.

RationalSearch implements Algorithm 7.3 on top of all four engines. The architecture of our extension is outlined in Figure 7.2. It intercepts PRISM's routine for solving constrained reachability probabilities and rewards, sharpening the probabilities every time it is invoked. These engines are built using floating point numbers, which can store at most 16 digits in the fractional part of the decimal expansion of any floating point number. Hence,

Figure 7.2 RationalSearch *architecture.* Given a PCTL formula φ, PRISM (equipped with CUDD) approximates the solution using value/interval iteration. The SHARPEN procedure uses this approximation $V^{\dagger}$ and employs findFraction, in conjunction with the rational extension to CUDD (CUDD + GMP), to generate a candidate rational vector. If this candidate rational vector satisfies an appropriate fixpoint check, it is guaranteed to be correct. Otherwise, the process is repeated with a better approximation.



the convergence criteria support a minimum ϵ of 10^{-16}. Our implementation, thus, bypasses the ϵ refinement loop from Algorithm 7.3 and directly invokes the procedure iteration for the maximum precision supported by doubles. Among the 4 engines, Explicit is the only one implemented entirely in Java. To support this engine, our tool uses the libraries JScience [145] and Apfloat [101] to construct the transition matrix using rational entries, perform matrix-vector multiplications for the fixpoint check in Algorithm 7.3, and implement the Kwek-Mehlhorn algorithm (Algorithm 7.1).

PRISM implements the remaining three engines using an extension of the CUDD library [146]. The off-the-shelf version of CUDD only supports floating point numbers at the terminals. RationalSearch enhances CUDD by allowing terminals to hold either floating points or arbitrary precision rational numbers provided by the GNU MP library [147]. Our extension allows the data type at a terminal node to be easily interchanged and the full suite of MTBDD operations can be performed regardless of the data type.

RationalSearch makes use of this extended CUDD functionality in the following manner. When the model is parsed, it constructs two transition matrices, one with doubles at the terminal nodes and one with rationals. The procedure iteration uses double-precision

transition matrix to generate a double-precision solution vector. RationalSearch translates this solution vector into a candidate solution vector stored as a rational MTBDDs using sharpen. The fixpoint check from sharpen can then be performed by an MTBDD matrix-vector multiplication between rational MTBDDs.

Algorithm 7.3 has also been integrated into the STORM model checker. Their implementation[2] differs from ours in that it supports running iteration with both floating-point and arbitrary-precision numbers. It begins by running value iteration using floating-point numbers and attempts to infer and exact solution from the approximation. If double-precision is determined to be insufficient for extracting the precise solution, the approximation engine is re-invoked using arbitrary-precision numbers. Another major difference in the STORM implementation is that STROM uses the Sylvan [148] MTBDD library instead of CUDD. Sylvan provides built-in support for arbitrary precision arithmetic.

7.5 EVALUATION

We evaluated our tool against examples involving quantitative reachability and rewards from the PRISM benchmark suite and case studies [129, 130] and compared the results with the exact parametric engines implemented in PRISM and STORM. In particular, we used version 4.3.1 of PRISM and version 1.0.0 of STORM. Our tests were carried out on an Intel core i7 dual core processor @2.2GHz with 8Gb RAM running macOS 10.12.4.

Performance overhead. We examined the overhead incurred by RationalSearch's extension of PRISM. The results are given in Table 7.1 for the approximation engines Explicit, MTBDD and Hybrid of PRISM. Due to the similarity between the Explicit and Sparse engines, we chose to only report metrics for the former. In Table 7.1, all of the tests were conducted using value iteration as the approximation scheme. The overhead incurred for interval iteration is similar and thus not reported.

On several examples with large state spaces, the Explicit engine fails with an out-of-memory exception. This can be attributed to the fact that the implementation stores two copies of the transition matrix in memory. On all the examples where Explicit fails, the symbolic engines (MTBDD and Hybrid) find the solution quickly, typically with an overhead of less than 50%. For the examples on which the Explicit engine did not encounter an out-of-memory exception, overhead times where much higher. One major reason for this difference is that the Explicit engine stores the solution vector as an array. Further, in this

[2]Information about the implementation of Algorithm 7.3 in STORM was obtain through private email conversations with the developers.

case, RationalSearch runs the sharpen procedure for each element of this array, thus resulting into redundant computation when a number appears multiple times. By contrast the symbolic engines perform symmetry reductions on the data structures and store only distinct values at the terminal nodes of the solution vector. As a result, sharpen needs only be run once for each terminal node.

Table 7.1 *Evaluation of* RationalSearch *overhead.* Columns 1-5 describe the benchmark examples. Columns 6-10 report the performance and overhead metrics for RationalSearch's extension of the various PRISM engines. Running times are reported in seconds. Overhead percentages were calculated by examining the time the routines added by RationalSearch contributed to the overall running time. All tests were conduced with the absolute convergence criterion ($\epsilon = 10^{-16}$), `javamaxmem=4g` and `cuddmaxmem=4g`. TO represents a timeout (set to 30 minutes), OOM indicates an out of memory exception and MP indicates that more than double precision is required to produce an exact answer. We write n/a if information could not be determined due to a timeout or an out of memory exception.

1	2	3	4	5	6	7	8	9	10	11
Model					Explicit		MTBDD		Hybrid	
Name	Type	Prop	Param	States	Time	Overhead	Time	Overhead	Time	Overhead
Biased Coins	DTMC	Reach	15	14348907	OOM	n/a	.18	62%	2.23	3%
IPv4	DTMC	Reach	100000	100003	4.1	254%	1708	1%	1702	1%
Crowds	DTMC	Reach	15	119800	MP	n/a	MP	n/a	MP	n/a
Lead. Elec.	DTMC	Cost	4	12302	1.5	117%	6.3	27%	19.6	7%
ECS	DTMC	PCTL	14	4815782	OOM	n/a	.4	70%	11.1	1%
Dice	MDP	Reach	6	4826809	OOM	n/a	.57	48%	2.4	6%
Din. Crypt.	MDP	Reach	9	855095	OOM	n/a	.381	41%	.84	13%
Fair Exch.	MDP	Reach	400	321600	11.2	480%	2.1	36%	2.07	37%
Firewire	MDP	Reach	11000	428364	87.7	640%	15.1	7%	16.7	7%
Din. Phil.	MDP	Cost	3	956	.54	55%	2.86	1%	.22	10%
Virus	MDP	Cost	3	809	.47	70%	2.3	1%	.2	19%
Dice Coin	MDP	PCTL	1	728	.76	260%	.168	17%	.13	22%

An encouraging observation from our results was that the overhead times did not vary drastically with the size of the model or the type of property being checked. In particular, both PCTL properties that we examined required solving three instances of constrained reachability properties. In spite of this, the overhead induced by RationalSearch on these examples remained consistent with the other examples.

Comparison with exact engines. We also compared RationalSearch with the exact engines implemented in PRISM and STORM. The results are reported in Table 7.2. The

existing exact engines of both PRISM and STORM were invoked with the `-exact` flag. In addition, STORM also uses the flag `--minmax:method pi`. RationalSearch was run with the underlying Hybrid engine and value iteration with absolute convergence criterion (with $\epsilon = 10^{-16}$) as the underlying approximation scheme. We set `javamaxmem=4g` and `cuddmaxmem=4g` wherever applicable.

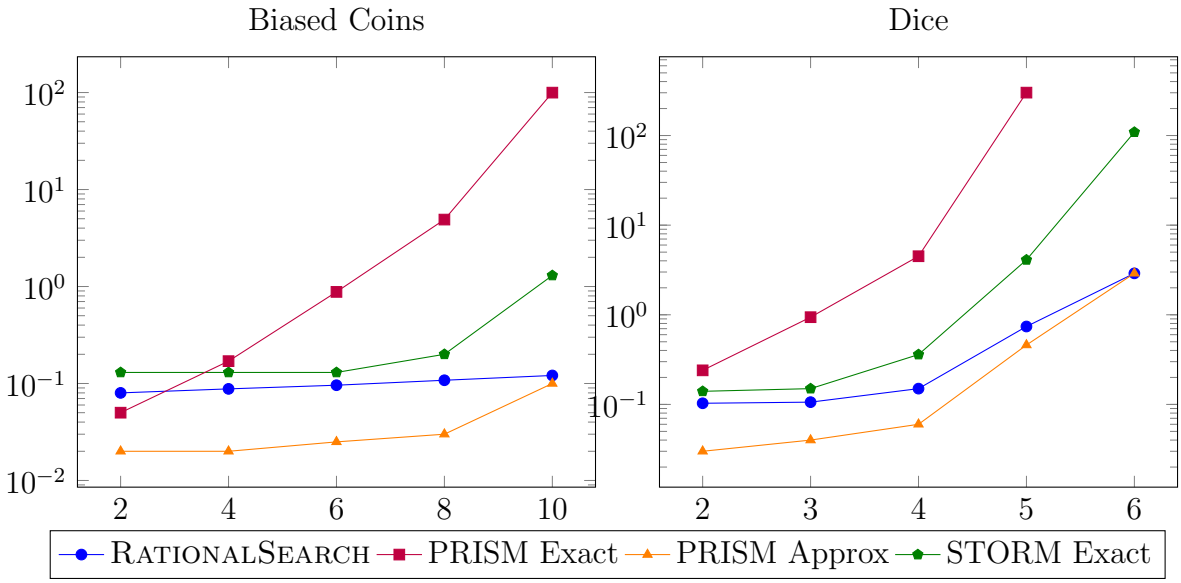
Table 7.2 *Experimental comparison of exact engines.* Columns 1-5 describe the benchmark examples. Columns 6,8,10 report the running times (in seconds) for each of the tools. Columns 7,9,11 report the portion of the model checking times (Columns 6,8,10) used for model construction. The configuration options for each of the tools is described in the main text. TO represents a timeout (set to 30 minutes) and OOM indicates an out of memory exception. We write n/a if information could not be determined due to a timeout or an out of memory exception. The PE in Columns 8 and 9 represent a parsing error in STORM.

1	2	3	4	5	6	7	8	9	10	11
Model					PRISM Exact		STORM Exact		RationalSearch	
Name	Type	Prop	Param	States	Time	Model	Time	Model	Time	Model
Biased Coins	DTMC	Reach	15	14348907	TO	n/a	458	375	2.23	.02
IPv4	DTMC	Reach	100000	100003	1141	6	342	.6	1702	1701
Lead. Elec.	DTMC	Cost	4	12302	70	1.7	1.37	0.2	19.6	1.2
ECS	DTMC	PCTL	14	4815782	TO	1435	TO	104	11.1	.04
Dice	MDP	Reach	6	4826809	TO	1016	109	76	2.4	.05
Din. Crypt.	MDP	Reach	9	855095	TO	39	12	11.5	.84	.06
Fair Exch.	MDP	Reach	400	321600	TO	2.7	.96	.05	2.07	1
Firewire	MDP	Reach	11000	428364	244	6.8	27	2.4	16.7	6.6
Din. Phil.	MDP	Cost	3	956	2.1	.2	.13	.125	.22	.03
Virus	MDP	Cost	3	809	1.3	.5	PE	PE	.2	.05
Dice Coin	MDP	PCTL	1	728	.12	.04	0.15	0.0	.13	.02

RationalSearch drastically outperformed PRISM's exact engine; in many cases, by several orders of magnitude. For about half of the examples, PRISM exact reached the 30 minute timeout. In every case, RationalSearch was able to find the exact solution in a matter of seconds. The comparison with STORM is more competitive. For the majority of the small and medium size examples (IPv4, Fair Exchange, Firewire, Dinning Philosophers) the running times for both engines was within the same order of magnitude. However, the performance benefit of RationalSearch became apparent with large models (Biased Coins, Dice, ECS). RationalSearch achieved a 200x speed-up on the biased coins example and 45x speed-up on the dice example. For the embedded control system example, RationalSearch returned a solution in a matter of seconds while both PRISM and STORM hit the 30 minute timeout.

In order to check the scalability of each of the exact engines, we also compared the running times on specific models (Biased Coins and Dice) where the number of states is governed by parameters that can be tuned to change the size of the underlying models. The results are depicted in Figure 7.3, where we use an approximate engine of PRISM as a baseline for our comparative analysis. Several interesting observations can be made here. As expected, the approximate engine of PRISM is the fastest. Since, RationalSearch is crucially tied to the approximate engine(s) in PRISM, it is not surprising again, that (RationalSearch) scales very well on large models, with comparable performance to the underlying approximate engine because of the low overhead our technique imposes. While the existing exact model checking engines in PRISM and STORM do perform well when the models are small, the performance quickly degrades when the models become reasonably large (the scale is a logarithmic scale). This clearly demonstrates the power of the insight that the approximate answers from fast iterative model checking techniques can be utilized to obtain exact rational solutions with only little overhead.

Figure 7.3 *Scaling comparison.* Running times for various model checking engines on the biased coins (left) and dice (right) examples. In both graphs, the values on the x-axis represent the parameters of the given model and the values on the y-axis represent the running times (in $\log_{10}$ scale). The configuration options for RationalSearch, PRISM Exact and STORM exact identical to those in Figure 7.2. PRISM approx was invoked using the same base options as RationalSearch. No data point is given for PRISM Exact with parameter 6 on the dice example as a 30 minute timeout was reached.



Comparison of iterative techniques. The final goal of our evaluation was to determine which approximation technique, amongst value iteration and interval iteration, could be more effectively integrated with Algorithm 7.3. In particular, we compared the two approaches for speed and the quality of their approximations. The results are given in Table 7.3. We integrated RationalSearch with the implementation of interval iteration in PRISM from prior work [124], available at [149].

To our surprise, we found that the interval iteration implementation from [149] did not always produce an approximate solution within the specified ϵ threshold. In particular, for the dice example under the parameter 6, the approximations for both $\epsilon = 10^{-6}$ and $\epsilon = 10^{-12}$ were not within the given threshold. This resulted in RationalSearch not being able infer an exact solution. Several other examples also suffered from this symptom. Although the approximate probabilities for the initial states were precise enough, poor approximations for the other states in the solution vector prevented RationalSearch from finding an exact solution.

The quality of solution produced by approximation techniques varied according to the ϵ threshold and the iterative technique used. Although we have not reported the numbers in Table 7.3, there are also examples for which the approximations for value (interval) iteration differ across the solution engines (for the same value of ϵ). In spite of this variation, RationalSearch is able to infer an exact solution for all of these different approximations.

In terms of speed, we observed only a small variance in the performance of the two techniques on the benchmarks we used. In most cases value iteration slightly outperformed interval iteration. The difference is primarily a result of the extra cost incurred by interval iteration to perform the additional pre-processing steps it requires. This cost outweighs the savings afforded by the version of shapren used with interval iteration that requires only a single fixpoint. In addition, our benchmarks did not identify any examples for which the improved precision of interval iteration allowed RationalSearch to infer an exact solution where value iteration could not. The preceding observations, in conjunction, lead us to conclude value iteration is the more effective partner for Algorithm 7.3.

Table 7.3 *Experimental comparison of iterative techniques.* Columns 1-5 describe the benchmark examples. Columns 6 and 9 are the approximate values generated by value iteration and interval iteration, respectively. Columns 8 and 10 report the running times for each engine (including the time for model construction). Column 7 gives the number of fixpoints checks computed by Algorithm 7.2. We do not report the number of fixpoint checks for interval iteration as the implementation of SHARPEN for this technique always calculates a single fixpoint. The probabilities given in columns 5,6 and 9 represent the probability of satisfying the given property from the initial state. The model types and properties for the evaluated examples are the same as in Figure 7.1. Both iterative techniques were invoke using the HYBRID engine with the options `javamaxmem=4g` and `cuddmaxmem=4g`. We write n/a in column 10 if no fixpoint was found by the SHAPREN procedure.

1	2	3	4	5	6	7	8	9	10
MODEL					VALUE ITERATION			INTERVAL ITERATION	
Name	Param	States	Epsilon	Solution	Approx	FP	Time	Approx	Time
Firewire	11000	428364	10^{-6}	2087481/2097152	0.9953885078430176	n/a	n/a	0.9953885078430176	n/a
Firewire	11000	428364	10^{-12}	2087481/2097152	0.9953885078430176	11	16.2	0.9953885078430176	27.7
Dice	3	2197	10^{-6}	1/216	0.004629455506801605	4	.1	0.004629705101251602	n/a
Dice	3	2197	10^{-12}	1/216	0.00462962962906488	4	.1	0.0046296296297008155	n/a
Dice	6	4826809	10^{-6}	1/46656	2.131238579750061E-5	n/a	n/a	2.143591779395712E-5	n/a
Dice	6	4826809	10^{-12}	1/46656	2.143347024102793E-5	9	2.6	2.1433470555450964E-5	n/a
Din. Crypt.	9	855095	10^{-6}	1/256	0.00390625	4	.71	0.00390625	.97
Din. Crypt.	9	855095	10^{-12}	1/256	0.00390625	4	1	0.00390625	1
Biased Coins	11	177147	10^{-6}	1/177147	5.645029269476758E-6	10	.11	5.6450292694767 58E-6	n/a
Biased Coins	11	177147	10^{-12}	1/177147	5.645029269476758E-6	10	.15	5.645029269476758E-6	.1
Din. Phil.	3	956	10^{-6}	27	26.999990834143837	1	.13	27.00000014876298	.28
Din. Phil.	3	956	10^{-12}	27	26.99999999999123	1	.14	27.000000000000142	.22
Lead. Elec.	4	12302	10^{-6}	256/49	5.2244897630362175	3	12.2	5.224489867467293	30.1
Lead. Elec.	4	12302	10^{-12}	256/49	5.224489795918261	3	12.4	5.22448979591833	29.7

CHAPTER 8: CONCLUSION

8.1 SUMMARY

In this work, we introduced a new framework for analyzing randomized security protocols against the powerful threat model of the Dolev-Yao attacker. In Chapter 4, we studied the problem of model checking safety and indistinguishability properties of randomized security protocols modeled using this new formalism. Our model checking algorithm assumed a bounded attacker that, at each protocol step, can construct a specified fixed number of inputs. We showed that the model checking problem for indistinguishability propertes in this context is in **PSPACE** and $\#\text{SAT}_d$-hard. Further, we proposed algorithms for deciding indistinguishability and safety properties in randomized security protocols and implemented our techniques in SPAN. The algorithms we developed are derived from techniques for analyzing POMDPs, objects that capture the semantics of randomized security protocols. In particular, we presented a POMDP indistinguishability algorithm by reducing the problem to PFA equivalence. This allowed us to establish that POMDP indistinguishability is in **NC**. We also provided algorithms for deciding safety and indistinguishability properties in POMDPs by utilizing translations to belief MDPs. While asymptotically less efficient than the PFA reduction algorithm, we showed, through experimental evaluation, that the later technique is more efficient in practice. As part of our evaluation, we modeled and analyzed several new classes of security protocols and discovered vulnerabilities in two electronic voting protocols.

In Chapters 5 and 6 we considered the problem of composition for randomized security protocols. Initially analyzing protocols with a bounded number of sessions, our composition result for indistinguishability properties considers indistinguishable protocols P and Q over equational theory E_a, and indistinguishable protocols P' and Q' over equational theory E_b. We showed that the composition of P and P' with Q and Q' preserves indistinguishability, provided E_a and E_b are disjoint. The same result applies to the case when both equational theories coincide and consist of symmetric encryption/decryption and hashes, provided each protocol message is tagged with a unique identifier for the protocol to which it belongs. Finally, we show that the latter result extends to protocols with an unbounded number of sessions, as long as messages from each session of the protocol are tagged with a unique session identifier.

Expanding on the previous results, we studied the problem of securely composing randomized security protocols under state-based safety properties. For one session, we show

that if P is secure with probability p and Q is secure with probability q then the composed protocol is secure with probability at least pq if the protocol messages are tagged with the information of which protocol they belong to. The same result applies to multiple sessions except that, in addition, the protocol messages of Q also need to be tagged with session identifiers.

In Chapter 7 we observed that techniques for exact model checking allow one to avoid logical errors in system analysis that can arise due to approximation techniques. We also observed that state-of-the-art exact quantitative model checkers don't scale as well as their approximate counterparts. To help overcome this limitation, we presented an algorithm and tool, RationalSearch, that computes the exact probabilities described by PCTL formulas for DTMCs and MDPs. Our tool works by sharpening approximate results obtained through iterative techniques, allowing it to benefit from the performance enhancements gained through approximation. Our experimental evaluation concurs with this hypothesis, and shows that our approach often performs significantly better than existing exact quantitative model checking tools while also scaling to large model sizes.

8.2 FUTURE DIRECTIONS

The techniques for symbolic analysis of randomized security protocols presented in this work lay the foundation for many exciting areas of follow-on research. Notice that the automated analysis techniques from Chapter 4 rely on an explicit exploration of the state space. As demonstrated by many state-of-the-art protocol analysis tools [1, 2], exploring the state space in a symbolic fashion can result in huge performance gains. One natural avenue for integrating symbolic analysis techniques into our model checking algorithms would be to group input recipes at a given protocol step according to how each one affects the distinguishability of executions in later branches of the protocol. For instance, consider the encoding of the mix-network from Example 2.3. Essentially, the security of the protocol is broken if there is some input recipe for the mix that makes the two possible probabilistic outputs distinguishable. If one could effectively identify (a symbolic pattern on) the recipes that make the outputs distinguishable, exploring many unnecessary portions of the state space could be avoided.

Another direction would be to explore different model checking approaches. For example, several protocol analysis engines [5, 38] use SAT/SMT solvers such as [150, 151] to help "guess" attacks. These techniques work by encoding the constraints of an attack as a logical formula and identifying vulnerabilities through the validity of such a formula. The challenge in using SAT based techniques in our context is encoding the parameters of a valid attack.

In the case of non-deterministic protocols, an attack is a sequence of legal protocol steps (including input recipes). By contrast, an attack on a randomized protocols is a tree with the following property. If two different input recipes are in two paths in the tree, than the executions encoded by those paths should be distinguishable. The latter is significantly harder to encode.

The speed of our model checking engine could also be enhanced using partial order reduction (POR) techniques. The idea behind POR is to classify protocol interleavings that induce "equivalent" behavior. In this group of equivalent interleavings, if there is a some interleaving that breaks a safety/indistinguishability property, then all interleavings in the group also break the respective property. Using such a classification, a protocol can be analyzed by exploring a single representative from each group of equivalent interleavings. POR reduction in randomized security protocols is likely to come with a synergy of the techniques for POR in non-deterministic security protocols [152, 153] and POR in probabilistic systems [154, 155].

Our model checking tool can also be used out-of-the-box to study many new classes of security protocols that were previously out of reach for symbolic verification. For example, mix networks are being used to enhance anonymity in blockchain protocols [156, 157, 158, 159, 160] and messaging systems [161, 162, 163]. Finally, there are several directions in which our composition results from Chapters 5 and 6 can be strengthened. As was done for non-randomized security protocols [31, 56], one can consider composing randomized security protocols that include dis-equality tests and equational theories with primitives beyond symmetric encryption/decryption and hashes.

APPENDIX A: AUXILIARY DEFINITIONS

Let P be a process. We will use $\mathsf{fv}(P)$ (resp. $\mathsf{bv}(P)$) to denote the set of variables that have some free (resp. bound) occurrence in P. As an auxiliary function needed to define the preceding, $\mathsf{abv}(P)$ will denote the set of variables for which every occurrence in P is bound.

$$\mathsf{bv}(P), \mathsf{abv}(P), \mathsf{fv}(P) = \begin{cases} \mathsf{bv}(P) = \mathsf{abv}(P) = \{x\}, & \text{if } P = \nu x \\ \mathsf{fv}(P) = \emptyset & \\ \\ \mathsf{bv}(P) = \mathsf{abv}(P) = \{x\}, & \text{if } P = (x := u) \\ \mathsf{fv}(P) = \mathsf{vars}(u) & \\ \\ \mathsf{bv}(P) = \mathsf{abv}(P) = \emptyset, & \text{if } P = [c_1 \wedge \ldots \wedge c_k] \\ \mathsf{fv}(P) = \mathsf{vars}(c_1) \cup \ldots \cup \mathsf{vars}(c_k) & \\ \\ \mathsf{bv}(P) = \mathsf{abv}(P) = \{x\}, & \text{if } P = \mathtt{in}(x) \\ \mathsf{fv}(P) = \emptyset & \\ \\ \mathsf{bv}(P) = \mathsf{abv}(P) = \emptyset, & \text{if } P = \mathtt{out}(u) \\ \mathsf{fv}(P) = \mathsf{vars}(u) & \\ \\ \mathsf{bv}(P_1) \cup \mathsf{bv}(P_2), & \text{if } P = P_1 \cdot P_2 \\ \mathsf{abv}(P_1) \cap \mathsf{abv}(P_2), & \\ \mathsf{fv}(P_1) \cup (\mathsf{fv}(P_2) \setminus \mathsf{abv}(P_1)) & \\ \\ \mathsf{bv}(P_1) \cup \mathsf{bv}(P_2), & \text{if } P = P_1 +_p P_2 \\ \mathsf{abv}(P_1) \cap \mathsf{abv}(P_2), & \\ \mathsf{fv}(P_1) \cup \mathsf{fv}(P_2) & \end{cases}$$

We can lift the preceding definition to a basic context D by requiring that $\mathsf{bv}(\Box) = \mathsf{abv}(\Box) = \mathsf{fv}(\Box) = \emptyset$ for any process variable $\Box$. Let C be a context of the form $B \cdot (D_1(\Box_1) \mid \ldots \mid D_m(\Box_m))$ where $B = a_1 \cdot \ldots \cdot a_n$. Define $\mathsf{bv}(C) = \mathsf{bv}(B) \cup \mathsf{bv}(D_1) \cup \ldots \cup \mathsf{bv}(D_m)$ and $\mathsf{fv}(C) = \mathsf{fv}(B) \cup ((\mathsf{fv}(D_1) \cup \ldots \cup \mathsf{fv}(D_m)) \setminus \mathsf{bv}(B))$.

REFERENCES

[1] B. Blanchet, M. Abadi, and C. Fournet, "Automated verification of selected equivalences for security protocols," *The Journal of Logic and Algebraic Programming*, vol. 75, no. 1, pp. 3–51, 2008.

[2] S. Escobar, C. Meadows, and J. Meseguer, "Maude-NPA: Cryptographic protocol analysis modulo equational properties," in *Foundations of Security Analysis and Design*. Springer, 2009, pp. 1–50.

[3] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Automated analysis of Diffie-Hellman protocols and advanced security properties," in *Computer Security Foundations*, 2012, pp. 78–94.

[4] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocol," *ACM Transactions on Computational Logic*, vol. 17, no. 4, 2016.

[5] A. Armando and L. Compagna, "SAT-based model-checking for security protocols analysis," *International Journal of Information Security*, vol. 7, no. 1, pp. 3–32, Jan 2008.

[6] M. Abadi and C. Fournet, "Mobile values, new names, and secure communication," in *Acm Sigplan Notices*, vol. 36, no. 3. ACM, 2001, pp. 104–115.

[7] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[8] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for web transactions," *ACM Transactions on Information and System Security*, vol. 1, no. 1, pp. 66–92, 1998.

[9] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 24, no. 2, pp. 84–90, 1981.

[10] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," in *Workshop on Information Hiding*, 1996, pp. 137–150.

[11] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," DTIC Document, Tech. Rep., 2004.

[12] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, "A fair protocol for signing contracts," *IEEE Transactions on Information Theory*, vol. 36, no. 1, pp. 40–46, 1990.

[13] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, vol. 28, no. 6, pp. 637–647, 1985.

[14] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider, and Z. Xia, "Prêt à voter: A voter-verifiable voting system," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 4, pp. 662–673, 2009.

[15] D. Chaum, P. Y. Ryan, and S. Schneider, "A practical voter-verifiable election scheme," in *European Symposium on Research in Computer Security*. Springer, 2005, pp. 118–139.

[16] B. Adida, "Helios: Web-based open-audit voting," in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[17] A. O. Santin, R. G. Costa, and C. A. Maziero, "A three-ballot-based secure electronic voting system," *Security and Privacy*, vol. 6, no. 3, pp. 14–21, 2008.

[18] C. A. Gunter, S. Khanna, K. Tan, and S. S. Venkatesh, "DoS protection for reliably authenticated broadcast," in *Network and Distributed System Security*, 2004.

[19] S. Delaune, S. Kremer, and M. Ryan, "Verifying privacy-type properties of electronic voting protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 435–487, 2009.

[20] C. Meadows, "Formal methods for cryptographic protocol analysis: Emerging issues and trends," *IEEE journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 44–54, 2003.

[21] C. Meadows, "Emerging issues and trends in formal methods in cryptographic protocol analysis: Twelve years later," in *Logic, Rewriting, and Concurrency*. Springer, 2015, pp. 475–492.

[22] L. de Alfaro, "The verification of probabilistic systems under memoryless partial-information policies is hard," Tech. Rep., 1999.

[23] L. Cheung, "Reconciling nondeterministic and probabilistic choices," Ph.D. dissertation, Radboud University of Nijmegen, 2006.

[24] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala, "Task-structured probabilistic I/O automata," in *Discrete Event Systems*, 2006.

[25] F. D. Garcia, P. Van Rossum, and A. Sokolova, "Probabilistic anonymity and admissible schedulers," *arXiv preprint arXiv:0706.1019*, 2007.

[26] K. Chatzikokolakis and C. Palamidessi, "Making random choices invisible to the scheduler," *Information and Computation*, 2010, to appear.

[27] R. Chadha, A. P. Sistla, and M. Viswanathan, "Model checking concurrent programs with nondeterminism and randomization," in *Foundations of Software Technology and Theoretical Computer Science*, 2010, pp. 364–375.

[28] M. Arapinis, T. Chothia, E. Ritter, and M. Ryan, "Analysing unlinkability and anonymity using the applied pi calculus," in *Computer Security Foundations*, 2010, pp. 107–121.

[29] D. Basin, J. Dreier, and R. Sasse, "Automated symbolic proofs of observational equivalence," in *Computer and Communications Security*, 2015, pp. 1144–1155.

[30] J. Dreier, C. Duménil, S. Kremer, and R. Sasse, "Beyond subterm-convergent equational theories in automated verification of stateful protocols," in *Principles of Security and Trust*, 2017.

[31] L. Hirschi, D. Baelde, and S. Delaune, "A method for verifying privacy-type properties: the unbounded case," in *Security and Privacy*, 2016, pp. 564–581.

[32] A. Armando, W. Arsac, T. Avanesov, M. Barletta, A. Calvi, A. Cappai, R. Carbone, Y. Chevalier, L. Compagna, J. Cuéllar et al., "The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2012, pp. 267–282.

[33] L. Viganò, "Automated security protocol analysis with the AVISPA tool," *Electronic Notes in Theoretical Computer Science*, vol. 155, pp. 61–86, 2006.

[34] C. J. Cremers, "The Scyther tool: Verification, falsification, and analysis of security protocols," in *Computer Aided Verification*. Springer, 2008, pp. 414–418.

[35] B. Blanchet, "Automatic verification of security protocols in the symbolic model: The verifier proverif," in *Foundations of Security Analysis and Design*. Springer, 2014, pp. 54–87.

[36] V. Cheval, "Apte: An algorithm for proving trace equivalence," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 587–592.

[37] V. Cheval, S. Kremer, and I. Rakotonirina, "DEEPSEC: Deciding equivalence properties in security protocols theory and practice," Ph.D. dissertation, INRIA Nancy, 2018.

[38] V. Cortier, A. Dallon, and S. Delaune, "SAT-Equiv: An efficient tool for equivalence properties," in *Computer Security Foundations*. IEEE, 2017, pp. 481–494.

[39] S. Kremer and M. Ryan, "Analysis of an electronic voting protocol in the applied pi calculus," in *European Symposium on Programming*. Springer, 2005, pp. 186–200.

[40] J. Goubault-Larrecq, C. Palamidessi, and A. Troina, "A probabilistic applied pi–calculus," in *Asian Symposium on Programming Languages and Systems*, 2007, pp. 175–190.

[41] D. Chaum, "The dining cryptographers problem: Unconditional sender and recipient untraceability," *Journal of Cryptology*, vol. 1, no. 1, pp. 65–75, 1988.

[42] R. L. Rivest, "The threeballot voting system," 2006.

[43] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer, 1992, pp. 244–251.

[44] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic, "A derivation system and compositional logic for security protocols," *Journal of Computer Security*, vol. 13, no. 3, pp. 423–482, 2005.

[45] C. He, M. Sundararajan, A. Datta, A. Derek, and J. C. Mitchell, "A modular correctness proof of IEEE 802.11i and TLS," in *ACM conference on Computer and Communications Security*, 2005, pp. 2–15.

[46] J. D. Guttman, "Authentication tests and disjoint encryption: A design method for security protocols," *Journal of Computer Security*, vol. 12, no. 3-4, pp. 409–433, 2004.

[47] J. D. Guttman, "Cryptographic protocol composition via the authentication tests," in *International Conference on Foundations of Software Science and Computational Structures*, 2009, pp. 303–317.

[48] S. Andova, C. J. F. Cremers, K. Gjøsteen, S. Mauw, S. F. Mjølsnes, and S. Radomirovic, "A framework for compositional verification of security protocols," *Information and Computation*, vol. 206, no. 2-4, pp. 425–459, 2008.

[49] V. Cortier, J. Delaitre, and S. Delaune, "Safely composing security protocols," in *Foundations of Software Technology and Theoretical Computer Science*, 2007, pp. 352–363.

[50] V. Cortier and S. Delaune, "Safely composing security protocols," *Formal Methods in System Design*, vol. 34, no. 1, pp. 1–36, 2009.

[51] M. Arapinis, S. Delaune, and S. Kremer, "From one session to many: Dynamic tags for security protocols," in *Logic for Programming, Artificial Intelligence, and Reasoning*, 2008, pp. 128–142.

[52] S. Delaune, S. Kremer, and M. D. Ryan, "Composition of password-based protocols," in *Computer Security Foundations*, 2008, pp. 239–251.

[53] C. Chevalier, S. Delaune, and S. Kremer, "Transforming password protocols to compose," in *Foundations of Software Technology and Theoretical Computer Science*, 2011, pp. 204–216.

[54] Ştefan Ciobâcă and V. Cortier, "Protocol composition for arbitrary primitives," in *Computer Security Foundations*, 2010, pp. 322–336.

[55] S. Mödersheim and L. Viganò, "Sufficient conditions for vertical composition of security protocols," in *ACM Symposium on Information, Computer and Communications Security*, 2014, pp. 435–446.

[56] M. Arapinis, V. Cheval, and S. Delaune, "Composing security protocols: From confidentiality to privacy," in *Principles of Security and Trust*, vol. 9036, 2015, pp. 324–343.

[57] M. Arapinis, V. Cheval, and S. Delaune, "Verifying privacy-type properties in a modular way," in *Computer Security Foundations*, 2012, pp. 95–109.

[58] M. Carbone and J. D. Guttman, "Sessions and separability in security protocols," in *Principles of Security and Trust*, 2013, pp. 267–286.

[59] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *Foundations of Computer Science*, 2001, pp. 136–145.

[60] R. Canetti and J. Herzog, "Universally composable symbolic analysis of mutual authentication and key-exchange protocols (extended abstract)," in *Theory of Cryptography*, 2006, pp. 380–403.

[61] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification*. Springer, 2011, pp. 585–591.

[62] C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk, "A storm is coming: A modern probabilistic model checker," in *Computer Aided Verification*. Springer, 2017, pp. 592–600.

[63] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, "Apex: An analyzer for open probabilistic programs," in *Computer Aided Verification*. Springer, 2012, pp. 693–698.

[64] V. Shmatikov, "Probabilistic analysis of anonymity," in *Computer Security Foundations*. IEEE, 2002, pp. 119–128.

[65] S. Schneider and A. Sidiropoulos, "CSP and anonymity," in *European Symposium on Research in Computer Security*, 1996, pp. 198–218.

[66] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.

[67] R. Chadha, A. P. Sistla, and M. Viswanathan, "Verification of randomized security protocols," in *Logic in Computer Science*. IEEE, 2017, pp. 1–12.

[68] M. Abadi and V. Cortier, "Deciding knowledge in security protocols under equational theories," *Theoretical Computer Science*, vol. 367, no. 1-2, pp. 2–32, 2006.

[69] M. Rusinowitch and M. Turuani, "Protocol insecurity with finite number of sessions is NP-complete," Ph.D. dissertation, INRIA, 2001.

[70] V. Cortier and S. Delaune, "A method for proving observational equivalence," in *Computer Security Foundations*, 2009, pp. 266–276.

[71] A. Serjantov, R. Dingledine, and P. Syverson, "From a trickle to a flood: Active attacks on several mix types," in *International Workshop on Information Hiding*. Springer, 2002, pp. 36–52.

[72] A. Serjantov and R. E. Newman, "On the anonymity of timed pool mixes," in *International Information Security Conference*. Springer, 2003, pp. 427–434.

[73] A. Serjantov and P. Sewell, "Passive attack analysis for connection-based anonymity systems," in *European Symposium on Research in Computer Security*. Springer, 2003, pp. 116–131.

[74] L. O'Connor, "On blending attacks for mixes with memory," in *Information Hiding*. Springer, 2005, pp. 39–52.

[75] C. Diaz and A. Serjantov, "Generalising mixes," in *Privacy Enhancing Technologies*. Springer, 2003, pp. 18–31.

[76] A. Serjantov, "A fresh look at the generalised mix framework," in *Privacy Enhancing Technologies*. Springer, 2007, pp. 17–29.

[77] C. Diaz and B. Preneel, "Taxonomy of mixes and dummy traffic," in *Information Security Management*. Springer, 2004, pp. 217–232.

[78] R. Dingledine, V. Shmatikov, and P. Syverson, "Synchronous batching: From cascades to free routes," in *Privacy Enhancing Technologies*. Springer, 2004, pp. 186–206.

[79] O. Berthold, A. Pfitzmann, and R. Standtke, "The disadvantages of free mix routes and how to overcome them," in *Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 30–45.

[80] P. Golle and A. Juels, "Dining cryptographers revisited," in *Theory and Applications of Cryptographic Techniques*. Springer, 2004, pp. 456–473.

[81] M. Naor, "Bit commitment using pseudorandomness," *Journal of Cryptology*, vol. 4, no. 2, pp. 151–158, 1991.

[82] D. Chaum, "Blind signatures for untraceable payments," in *Advances in Cryptology*. Springer, 1983, pp. 199–203.

[83] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct randolli functions," in *Foundations of Computer Science*. IEEE, 1984, pp. 464–479.

[84] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, "On the complexity of the equivalence problem for probabilistic automata," in *International Conference on Foundations of Software Science and Computational Structures*. Springer, 2012, pp. 467–481.

[85] L. Doyen, T. A. Henzinger, and J.-F. Raskin, "Equivalence of labeled Markov chains," *Foundations of Computer Science*, vol. 19, no. 03, pp. 549–563, 2008.

[86] R. Lenhardt, J. Worrell, and J. Ouaknine, "Probabilistic automata with parameters," 2009.

[87] W.-G. Tzeng, "A polynomial-time algorithm for the equivalence of probabilistic automata," *SIAM Journal on Computing*, vol. 21, no. 2, pp. 216–227, 1992.

[88] P. S. Castro, P. Panangaden, and D. Precup, "Equivalence relations in fully and partially observable Markov decision processes," in *International Joint Conference on Artificial Intelligence*, vol. 9, 2009, pp. 1653–1658.

[89] K. Chatterjee, M. Chmelík, and M. Tracol, "What is decidable about partially observable Markov decision processes with omega-regular objectives," *Journal of Computer and System Sciences*, vol. 82, no. 5, pp. 878 – 911, 2016.

[90] D. Braziunas, "POMDP solution methods," *University of Toronto*, 2003.

[91] A. R. Cassandra, "A survey of POMDP applications," in *Working notes of AAAI 1998 fall symposium on planning with partially observable Markov decision processes*, vol. 1724, 1998.

[92] G. Norman, D. Parker, and X. Zou, "Verification and control of partially observable probabilistic systems," *Real-Time Systems*, vol. 53, no. 3, pp. 354–402, May 2017.

[93] A. Paz, *Introduction to probabilistic automata*. Academic Press, 1971.

[94] "SPAN," https://github.com/bauer-matthews/SPAN.

[95] M. Abadi and V. Cortier, "Deciding knowledge in security protocols under equational theories," *Theoretical Computer Science*, vol. 367, no. 1, pp. 2 – 32, 2006, automated Reasoning for Security Protocol Analysis. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030439750600572X

[96] R. Chadha, V. Cheval, Ş. Ciobâcă, and S. Kremer, "Automated verification of equivalence properties of cryptographic protocols," *ACM Transactions on Computational Logic*, vol. 17, no. 4, p. 23, 2016.

[97] P. Narendran, F. Pfenning, and R. Statman, "On the unification problem for cartesian closed categories," in *Logic in Computer Science*. IEEE, 1993, pp. 57–63.

[98] B. Conchinha, D. A. Basin, and C. Caleiro, "FAST: An efficient decision procedure for deduction and static equivalence," in *International Conference on Rewriting Techniques and Applications*, 2011, p. 11.

[99] J.-P. Jouannaud, "Solving equations in abstract algebras: A rule-based survey of unification," in *Computational Logic*. MIT-Press, 1991, pp. 257–321.

[100] "Google Guava," https://github.com/google/guava.

[101] "Apfloat," http://www.apfloat.org/.

[102] "Graphviz," https://www.graphviz.org/.

[103] "Span indistinguishability examples," https://github.com/bauer-matthews/SPAN/tree/master/src/test/resources/examples/indistinguishability.

[104] "Span reachability examples," https://github.com/bauer-matthews/SPAN/tree/master/src/test/resources/examples/reachability.

[105] B. Pfitzmann and A. Pfitzmann, "How to break the direct RSA-implementation of mixes," in *Theory and Application of of Cryptographic Techniques*, 1989, pp. 373–381.

[106] S. Ciobâca, "Verification and composition of security protocols with applications to electronic voting," Ph.D. dissertation, ENS Cachan, 2012.

[107] E. W. Dijkstra, "Self-stabilization in spite of distributed control," in *Selected Writings on Computing: a Personal Perspective*. Springer, 1982.

[108] M. Rabin, "Randomized Byzantine generals," in *Foundations of Computer Science*, 1983, pp. 403–409.

[109] M. Duflot, M. Kwiatkowska, G. Norman, and D. Parker, "A formal analysis of bluetooth device discovery," *Software Tools for Technology Transfer*, vol. 8, no. 6, pp. 621–632, 2006.

[110] D. Bhaduri, S. K. Shukla, P. S. Graham, and M. B. Gokhale, "Reliability analysis of large circuits using scalable techniques and tools," *IEEE Transactions on Circuits and Systems*, vol. 54, 2007.

[111] J. Han, H. Chen, E. Boykin, and J. Fortes, "Reliability evaluation of logic circuits using probabilistic gate models," *Microelectronics Reliability*, 2011.

[112] N. Mohyuddin, E. Pakbaznia, and M. Pedram, "Probabilistic error propagation in a logic circuit using the boolean difference calculus," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*. Springer, 2011, pp. 359–381.

[113] G. Norman, D. Parker, M. Kwiatkowska, and S. Shukla, "Evaluating the reliability of NAND multiplexing with PRISM," *Computer-Aided Design of Integrated Circuits and Systems*, 2005.

[114] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *Computer-Aided Design of Integrated Circuits and Systems*, 1999.

[115] Q. Qiu, Q. Qu, and M. Pedram, "Stochastic modeling of a power-managed system-construction and optimization," *Computer-Aided Design of Integrated Circuits and Systems*, 2001.

[116] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol," *Formal Aspects of Computing*, vol. 14, no. 3, pp. 295–318, 2003.

[117] M. Kwiatkowska, G. Norman, and J. Sproston, "Probabilistic model checking of the IEEE 802.11 wireless local area network protocol," in *Process Algebra and Probabilistic Methods*, 2002.

[118] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (eds.), ser. CRM Monograph Series. American Mathematical Society, 2004, vol. 23.

[119] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[120] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám, "Prophesy: A probabilistic parameter synthesis tool," in *Computer Aided Verification.* Springer, 2015, pp. 214–231.

[121] P. R. D'argenio and K. G. Larsen, "Rapture: A tool for verifying Markov decision processes," in *Conference on Concurrency Theory*, 2002.

[122] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang, "PARAM: A model checker for parametric Markov models," in *Computer Aided Verification*, 2010.

[123] J.-P. Katoen, M. Khattri, and I. Zapreevt, "A Markov reward model checker," in *Quantitative Evaluation of Systems.* IEEE, 2005.

[124] C. Baier, J. Klein, L. Leuschner, D. Parker, and S. Wunderlich, "Ensuring the reliability of your model checker: Interval iteration for markov decision processes," in *Computer Aided Verification*, 2017.

[125] S. Haddad and B. Monmege, "Reachability in MDPs: Refining convergence of value iteration," in *International Workshop on Reachability Problems.* Springer, 2014, pp. 125–137.

[126] R. Wimmer, A. Kortus, M. Herbstritt, and B. Becker, "Probabilistic model checking and reliability of results," in *Design and Diagnostics of Electronic Circuits and Systems.* IEEE, 2008, pp. 1–6.

[127] R. St-Aubin, J. Hoey, and C. Boutilier, "APRICODD: Approximate policy construction using decision diagrams," in *Advances in Neural Information Processing Systems*, 2001, pp. 1089–1095.

[128] S. Kwek and K. Mehlhorn, "Optimal search for rationals," *Information Processing Letters*, vol. 86, no. 1, pp. 23–26, 2003.

[129] "PRISM Benchmark Suite," http://www.prismmodelchecker.org/benchmarks/.

[130] "PRISM Case Studies," http://www.prismmodelchecker.org/casestudies/.

[131] S. Giro, "Efficient computation of exact solutions for quantitative model checking," in *Quantitative Aspects of Programming Languages*, 2012.

[132] C. Daws, "Symbolic and parametric model checking of discrete-time Markov chains," in *International Colloquium on Theoretical Aspects of Computing.* Springer, 2004, pp. 280–294.

[133] E. M. Hahn, T. Han, and L. Zhang, "Synthesis for PCTL in parametric markov decision processes," in *NASA Formal Methods Symposium*. Springer, 2011, pp. 146–161.

[134] E. M. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric Markov models," *International Journal on Software Tools for Technology Transfer*, vol. 13, no. 1, pp. 3–19, 2011.

[135] M. Kwiatkowska, G. Norman, and D. Parker, "Controller dependability analysis by probabilistic model checking," in *Information Control Problems in Manufacturing*, 2004.

[136] J. J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker, *Mathematical techniques for analyzing concurrent and probabilistic systems*. American Mathematical Society, 2004.

[137] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated verification techniques for probabilistic systems," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Springer, 2011, pp. 53–113.

[138] M. Fujita, P. C. McGeer, and J.-Y. Yang, "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation," *Formal Methods in System Design*, vol. 10, no. 2-3, pp. 149–169, 1997.

[139] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier, "SPUDD: Stochastic planning using decision diagrams," in *Uncertainty in Artificial Intelligence*, 1999.

[140] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 100, no. 8, 1986.

[141] K. L. McMillan, *Symbolic Model Checking*. Norwell, MA, USA: Kluwer Academic Publishers, 1993.

[142] J. E. Hopcroft, *Introduction to automata theory, languages, and computation*. Pearson Education India, 2008.

[143] "RationalSearch," https://publish.illinois.edu/rationalmodelchecker/.

[144] D. Parker, "Implementation of symbolic model checking for probabilistic systems," Ph.D. dissertation, University of Birmingham, 2002.

[145] "JScience," http://jscience.org/.

[146] "CUDD," http://vlsi.colorado.edu/~fabio/CUDD/html/.

[147] "GNU multiple precision arithmetic library," https://gmplib.org/.

[148] T. van Dijk and J. van de Pol, "Sylvan: Multi-core decision diagrams," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2015, pp. 677–691.

[149] "Ensuring the Reliability of Your Model Checker: Interval Iteration for Markov Decision Processes," https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/CAV17/.

[150] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[151] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Computer Aided Verification*. Springer, 2011, pp. 171–177.

[152] E. Clarke, S. Jha, and W. Marrero, "Partial order reductions for security protocol verification," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2000, pp. 503–518.

[153] W. Fokkink, M. T. Dashti, and A. Wijs, "Partial order reduction for branching security protocols," in *Application of Concurrency to System Design*. IEEE, 2010, pp. 191–200.

[154] C. Baier, M. Größer, and F. Ciesinski, "Partial order reduction for probabilistic systems," in *Quantitative Evaluation of Systems*, vol. 4, 2004, pp. 230–239.

[155] M. Groesser and C. Baier, "Partial order reduction for Markov decision processes: A survey," in *International Symposium on Formal Methods for Components and Objects*. Springer, 2005, pp. 408–427.

[156] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten, "Mixcoin: Anonymity for bitcoin with accountable mixes," in *International Conference on Financial Cryptography and Data Security*. Springer, 2014, pp. 486–504.

[157] T. Ruffing, P. Moreno-Sanchez, and A. Kate, "CoinShuffle: Practical decentralized coin mixing for Bitcoin," in *European Symposium on Research in Computer Security*. Springer, 2014, pp. 345–364.

[158] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore, "Sybil-resistant mixing for bitcoin," in *Workshop on Privacy in the Electronic Society*. ACM, 2014, pp. 149–158.

[159] L. Valenta and B. Rowan, "Blindcoin: Blinded, accountable mixes for bitcoin," in *Financial Cryptography and Data Security*. Springer, 2015, pp. 112–126.

[160] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle, "Coinparty: Secure multi-party mixing of bitcoins," in *Data and Application Security and Privacy*. ACM, 2015, pp. 75–86.

[161] A. M. Piotrowska, J. Hayes, T. Elahi, S. Meiser, and G. Danezis, "The loopix anonymity system," in *USENIX Security Symposium*, 2017, pp. 16–18.

[162] N. Tyagi, Y. Gilad, D. Leung, M. Zaharia, and N. Zeldovich, "Stadium: A distributed metadata-private messaging system," in *Symposium on Operating Systems Principles*. ACM, 2017, pp. 423–440.

[163] A. Kwon, H. Corrigan-Gibbs, S. Devadas, and B. Ford, "Atom: Horizontally scaling strong anonymity," in *Symposium on Operating Systems Principles*. ACM, 2017, pp. 406–422.